Tutoriel sur dcm4che2

Par dimitri PIANETA

Version initiale 2009

Version finale 2016

Table des matières

I) Historique :

Dans les années 2000, Gunter ZEILINGER avait écrit JDicomutility suite utilisé par le Soflink (maintenant TriSpark) Java DICOM Toolkit (JDT). Après cette expérience avec le JDT, il décidait décrire un package DICOM toolkit. Ainsi est née dcm4che (prononcé d-c-m-for-chay).

Le site internet est le suivant : www.dcm4che.org

II) Description du package :

Je vais faire un petit tour des packages utiles pour la programmation en java.

- ***org.dcm4che2.audit.log4j*** : permet de gérer les erreurs du package.

Class Summary :

AuditMessageFilter : est la classe qui permet de faire le lien internet avec différent méthode.

- ***org.dcm4che2.audit.log4j.helpers :***

Class Summary:

SyslogWriter : SyslogWriter is a wrapper around the java.net.DatagramSocket class so that it behaves like a java.io.Writer.

- ***org.dcm4che2.audit.message*** :

Provides classes for generating XML formated Audit Messages compliant to the "Audit Trail and Node Authentication (ATNA) Integration Profile" specified in the IHE IT Infrastructure Technical Framework, which itself utilizes RFC 3881 - Security Audit and Access Accountability Message XML Data Definitions for Healthcare Applications and its extension by DICOM Supplement 95: Audit Trail Messages.

- ***org.dcm4che2.image :*** *Package de la gestion des caractéristiques de l'image*

Class Summary	
ByteLookupTable	
ColorModelFactory	
LookupTable	
OverlayUtils	Provides utility methods to extract overlay information from DICOM files.
ShortLookupTable	
SimpleYBRColorSpace	
VOIUtils	

- ***org.dcm4che2.imageio*** : *Package de la gestion des caractéristiques de lecture des données de l'image*

Class Summary	
ImageReaderFactory	
ImageWriterFactory	
ItemParser	
ItemParser.Item	

- ***org.dcm4che2.imageio.plugins.dcm***

Class Summary	
DicomImageReadParam	
DicomStreamMetaData	

- ***org.dcm4che2.imageioimpl.plugins.dcm***

Class Summary	
DicomImageReader	
DicomImageReaderSpi	
DicomImageWriter	Write DICOM files containing images - handles transcoding to a new representation, as well as transcoding the images.
DicomImageWriterSpi	Provide information about the dicom image writers

- ***org.dcm4che2.io***

Class Summary	
ContentHandlerAdapter	
DataOutputStreamAdapter	This class adapts a regular output stream onto a data output stream, such as would be found in an ImageOutputStream
DicomInputStream	
DicomOutputStream	
ImageInputStreamAdapter	
SAXReader	
SAXWriter	
StopTagInputHandler	
TranscoderInputHandler	

- ***org.dcm4che2.iod.module :*** *Donne différent module utile des valeurs de tag et de caractéristique pour les images Dicom*

Class Summary	
Composite	
CRImage	
DXImage	The Digital X-Ray (DX) Image Information Object Definition specifies an image that has been created by a digital projection radiography imaging device.
Image	
SpatialFiducials	The Fiducials IOD specifies the spatial relationship between the Composite Fiducial instance, to one or more images.

- ***org.dcm4che2.iod.module.composite***

This package contains classes which represent the Modules that are common to all Composite Image IODs (C.7 COMMON COMPOSITE IMAGE IOD MODULES).

Package org.dcm4che2.image

ByteLookupTable	@return: inBits, signed, off, outBits, data
ColorModelFactory	Classe qui permet de créer le model color
LookupTable	Création de la lookupTable
OverlayUtils	Provides utility methods to extract overlay information from DICOM files.
PartialComponentColorModel	Donne les couleurs de base
PartialComponentSampleModel	Sub-sample x,y are the rate of sub-sampling.
ShortLookupTable	
SimpleYBRColorSpace	Paramètre YBR
VOIUtils	Différents paramètres de calcul VOI

Je décris que les méthodes de chaque classe.

Class ByteLookupTable		
Retour	**Méthode**	**Explication**
Int	length()	@return la taille d'un tableau
byte	lookupByte(int in)	Recherche les bits pour un entier
short	lookupShort(int in)	@return lookupByte(in) & 0xff
int	lookup(int in)	@return lookupByte(in) & 0xff
byte[]	lookup(byte[] src, int srcPos, byte[] dst, int dstPos, int length, int channels, int skip)	
short[]	Lookup(byte[] src, int srcPos, short[] dst, int dstPos, int length)	
int[]	Lookup(byte[] src, int srcPos, int[] dst, int dstPos, int length, int alpha)	
byte[]	Lookup(short[] src, int srcPos, byte[] dst, int dstPos, int length)	
int[]	Lookup(short[] src, int[] dst, int dstPos, int length, int alpha)	
LookupTable	scale(int outBits, Boolean inverse, short[] pval2out)	L'échelle du tableau
LookupTable	combine(LookupTable other, int outBits, Boolean inver, short[] pval2out)	
LookupTable	combine(LookupTable vlut, LookupTable plut, int outBits, Boolean inverse, short[] pval2out)	

Class ColorModelFactory		
Retour	**Méthode**	**Explication**
Java.awt.image.ColorModel	createColorModel(DicomObject s)	Création de la ColorModel
boolean	isMonochrome(DicomObject ds)	Comparaison entre le tag SamplesPerPixel et les couleurs de la palette appartient à la photomérie
Java.awt.image.ColorModel	createPaletteColorModel(DicomObject ds)	@return IndexColorModel(int

		bits, int size, byte[] r, byte[] g, byte[] b)

Class LookupTable		
Retour	**Méthode**	**Explication**
LookupTable	combine(LookupTable other, int outBits, Boolean inverse, short[] pval2out)	Création de la ColorModel
LookupTable	combine(LookupTable vlut, LookupTable plut, int outBits, Boolean inverse, short[] pval2out)	Create LUT for given i/o range, non-linear Modality LUT and non-linear VOILUT.
LookupTable	createLut(int inBits, Boolean signed, int outBits, DicomObject mLut, DicomObject voilut, Boolean inverse, short[] pval2out)	Create LUT for given i/o range, non-linear Modality LUT and Window Center/Width.
LookupTable	createLut(int inBits, Boolean signed, int outBits, DicomObject mLut, DicomObject voilut, DicomObject pLut, Boolean inverse, short[] pval2out)	Create LUT for given i/o range, Rescale Slope/Intercept, non-linear VOILUT and non-linear Presentation LUT.
LookupTable	createLut(int inBits, Boolean signed, int outBits, DicomObject mLut, float center, float width, java.lang.String vlutFct, boolean inverse, short[] pval2out)	Create LUT for given i/o range, non-linear Modality LUT and Window Center/Width
LookupTable	createLut(int inBits, Boolean signed, int outBits, DicomObject mLut, float center, float width, java.lang.String vlutFct, DicomObject pLut, Boolean inverse, short[] pval2out)	Create LUT for given i/o range, non-linear Modality LUT, Window Center/Width and non-linear Presentation LUT
LookupTable	createLut(int in Bits, Boolean signed, int outBits, float slope, float intercept, DicomObject voiLut, Boolean inverse, short[] pval2out)	Create LUT for given i/o range, Rescale Slope/Intercept and non-linear VOI LUT.
LookupTable	CreateLut(int inBits, Boolean signed, int outBits, float slope, float intercept, DicomObject voiLut, DicomObject pLut, Boolean inverse, short[] pval2out)	Create LUT for given i/o range, Rescale Slope/Intercept, non-linear VOI LUT and non-linear Presentation LUT

Class LookupTable		
Retour	**Méthode**	**Explication**
LookupTable	**createLut**(int inBits, Boolean signed, int outBits, float slope, float intercept, float center, float width, java.lang.String vlutFct, Boolean inverse, short[] pval2out)	Create ramp or sigmoid LUT for given i/o range, Rescale Slope/Intercept and Window Center/Width.
LookupTable	**createLut**(int inBits, Boolean signed, int outBits, float slope, float intercept, float center, float width, java.lang.String vlutFct, Boolean inverse, short[] pval2out)	Create LUT for given i/o range, Rescale Slope/Intercept, Window Center/Width and non-linear Presentation LUT.
LookupTable	**createLutForImage**(DicomObject img, DicomObject mlutObj, DicomObject voiObj, DicomObject pLut, float center, float width, java.lang.String vlutFct, short[] pval2out)	Create LUT for given DICOM image and output range.
LookupTable	**createLutForImage**(DicomObject img, int outBits, short[] pval2out)	Create LUT for given DICOM image and output range.
LookupTable	**createLutForImageWithPR**(DicomObject img, DicomObject mLut, DicomObject voiLut, boolean inverse, int outBits, short[] pval2out)	Create LUT for given DICOM image, non-linear VOI LUT and output range.
LookupTable	**createLutFromWL**(DicomObject img, DicomObject mLut, float center, float width, java.lang.String vlutFct, boolean inverse, int outBits, short[] pval2out)	Create LUT for given DICOM image, modality LUT, and Window Center/Width and output range.
int	getOffset	@return off
int	length()	@ return la taille d'un tableau
lookupTable	scale(int outBits, Boolean inverse, short[] pval2out)	
Int	toIndex(int in)	.

Class OverlayUtils		
Retour	**Méthode**	**Explication**
Int	**extractFrameNumber**(int imageIndex)	Extra the frame number portion of the overlay number/imageIndex value.
Java.awt.image.BufferedImage	**extractOverlay**(DicomObject ds, int overlayNumber, javax.imageio.ImageReader reader, java.lang.String rgbs)	Read an overlay image or region instead of a regular image.
int	**getOverlayHeight**(DicomObject ds, int overlayNumber)	Reads the height of the overlay - needs to be done separately from the primary width even though they are supposed to be identical, as a stand alone overlay won't have any width/height except in the overlay tags.
boolean	**getOverlayWidth**(DicomObject ds, int overlayNumber)	Reads the width of the overlay - needs to be done separately from the primary width even though they are supposed to be identical, as a stand alone overlay won't have any width/height except in the overlay tags.
boolean	**isOverlay**(int imageIndex)	Returns true if the given frame number references an overlay - that is, is the form 0x60xx yyyy where xx is the overlay number, and yyyy is the overlay frame number.

Class VOIUtils		
Retour	**Méthode**	**Explication**
Int[]	calcMinMax(DicomObject img, java.awt.image.Raster raster)	Gets the min/max value from a data buffer, in the raw pixel data.
boolean	containsVOIAttributes(DicomObject dobj)	Return true if the specified object contains some type of VOI attributes at the current level
Float[]	getMinMaxWindowCenterWidth(DicomObject img, DicomObject pr, int frame, java.awt.iamge.Raster raster)	This method returns the minimum and maximum window center widths, based on the Modality LUT and image information.
boolean	isModalityLUTcontainsPixelIntensityRelationshipLUT(DicomObject img)	Return true if the specifield image object contains a pixel intensity realationship LUT, based on SOP class
boolean	isModalityLUTcontainsPixelIntensityRelationshipLUT(java.lang.String uid)	Return true if the specifield image object contains a pixel intensity relationship LUT
DicomObject	selectModalityLUTObject(DicomObject img, DicomObject pr, int frame)	Finds the applicable DicomObject containing the Modality LUT information for given frame within the image.
DicomObject	selectVoiItemFromPr(java.lang.String iuid, DicomObject pr, int Frame)	Searches for a Scoftcopy VOILUT Sequence for the given frame, or one for just the SOP instance
DicomObject	selectVoiObject(DicomObject img, DicomObject pr, int frame)	Finds the application DicomObject containing the VOI LUT information for the given frame within the image

Class SimpleYBRColorSpace		
Retour	**Méthode**	**Explication**
Java.awt.color.ColorSpace	createYBRFullColorSpace(java.awt .color.ColorSpace rgbCS)	Créer les composantes YBR qui donne le Full Color Space.
Java.awt.color.ColorSpace	createYBRPartialColorSpace(java. awt.color.ColorSpace rgbCs)	Créer les composantes YBR qui dpnne le Patial Color Space.
boolean	Equals(java.lang.Object o)	Comparaison entre un ffichier entrée et les couleurs de bases.
Float[]	fromCIEXYZ(float[] xyzvalue)	
Float[]	fromRGB(float[] rgb)	@return new float[] {y, cb, cr}
void	main(java.lang.String[] args)	Convertie TO_YBR_FULL et TO_YBR_PARTIAL en YBR.
Float[]	toCIEXYZ(float[] colorValue)	
Float[]	toRGB(float[] ybr)	@return la conversion ybr.

Class PartialComponentColorModel		
Retour	**Méthode**	**Explication**
Java.awt.image.SampleModel	**createCompatibleSample**(`int w , int h)`	Créer
int	**getAlpha**(int pixel)	Obtient alpha d'un pixel @return 255
int	**getAlpha**(`java.lang.Object inData)`	Obtient alpha d'un object pixel @return 255
int	**getBlue**(int pixel)	Obtient la couleur bleu d'un pixel @return pixel & 0xFF.
int	**getBlue**(`java.lang.Object inData)`	@return new float[] {y, cb, cr}
int	**getGreen**(int pixel)	Obtient la couleur vert d'un pixel @return pixel & 0xFF.

int	**getGreen**(`java.lang.Object inData`)	@return (getRGB(inData) >> 8) & 0xFF;
int	**getRed(**int pixel)	Obtient la couleur rouge d'un pixel @return pixel & 0xFF0000.
int	**getRed**(`java.lang.Object inData`)	@return getRGB(inData) >> 16;
int	**getRGB**(`java.lang.Object inData`)	@return ret = (((int) (rgb[0] * 255)) << 16) | (((int) (rgb[1] * 255)) << 8) | (((int) (rgb[2] * 255))
boolean	isCompatibleRaster(java.awt.image.Raster raster)	

Class PartialComponentSampleModel		
Retour	**Méthode**	**Explication**
Java.awt.image.SampleModel	createCompatibleSampleModel(int w, int h)	
Java.awt.image.DataBuffer	**createDataBuffe**r()	
Java.awt.image.SampleModel	createSubsetSampleModel(int[] bands)	
Java.lang.Object	getDataElements(int x, int y, java.lang.Object obj, java.awt.image.DataBuffer data)	
int	**getNumDataElements**()	
int	**getSample**(int x, int y, int b, java.awt.image.DataBuffer data)	
Int[]	**getSampleSize**()	
int	**getSampleSize**(int band)	

void	**setDataElements**(int x, int y, java.lang.Object obj, java.awt.image.DataBuffer data)	
void	**setSample**(int x, int y, int b, int s, java.awt.image.DataBuffer data)	

Package org.dcm4che2.imageio

ImageReaderFactory	
ImageWriterFactory	
ItemParser	
ItemParser.Item	

Je décris que les méthodes de chaque classe.

Class ImageReaderFactory

Retour	Méthode	Explication
void	**adjustDatasetForTransfertSyntax**(Dicom Object ds, java.lang.String tsuid)	Adaptation de l'image en carctèristique
ImageReaderFactory	**getInstance()**	@return un nouveau ImageReaderFactory
Javax.imageio.ImageReader	**getReaderForTransfertSyntax(**java.lang.String tsuid)	Obtient un ImageReader (lire les pixels) en vérifiant le bon format.
boolean	**needsImageTypeSpecifier**(java.lang.String tsuid)	Some image types need an image type specifier in order to figure out the source image information - if that is the case, then this will return true, based on configuration [tsuid].typeSpecifier=true in the config file.

Class ImageWriterFactory

Retour	Méthode	Explication
Javax.imageio.ImageWriteParam	**createWriteParam(**java.lang.String tsuid, javax.imageio.ImageWriter writer)	This is used to create a param for writing the sub-image.
ImageWriterFactory	**getInstance()**	@return un nouveau ImageWriterFactory
Javax.imageio.ImageWritter	**getWritterForTransfertSyntax(**java.lang.String tsuid)	Obtient un ImageWriter (écrit pixels) en vérifiant le bon format.

Class ItemParser		
Retour	**Méthode**	**Explication**
int	**getNumberOfDataFragments()**	This is used to create a param for writing the sub-image.
com.sun.media.imageio.stream.StreamSegment	**getIStreamSegment**(long pos, int len**)**	
void	**getIStreamSegment** (long pos, int len, com.sun.media.imageio.stream.StreamSegment seg)	

Class ItemParser(suite)		
Retour	**Méthode**	**Explication**
int	**getNumberOfDataFragments()**	
com.sun.media.imageio.stream.StreamSegment	**getIStreamSegment**(long pos, int len**)**	
void	**getIStreamSegment** (long pos, int len, com.sun.media.imageio.stream.StreamSegment seg)	
byte[]	**readFrame**(com.sun.media.imageio.stream.SegmentedImageInputStream siis, int frame)	Lit les octets pixels et transformation en octets.
void	**seekFooter()**	Recherche l'Item
void		

	seekFrame(com.sun.media.imageio.stream.SegmentedImageInputStream siis, int frame)	

On définit la liste des formats internes de l'image dicom peut prendre :

UID.JPEGBaseline1, UID.JPEGExtended24,

UID.JPEGExtended35Retired,

UID.JPEGSpectralSelectionNonHierarchical68Retired,

UID.JPEGSpectralSelectionNonHierarchical79Retired,

UID.JPEGFullProgressionNonHierarchical1012Retired,

UID.JPEGFullProgressionNonHierarchical1113Retired,

UID.JPEGLosslessNonHierarchical14,

UID.JPEGLosslessNonHierarchical15Retired,

UID.JPEGExtendedHierarchical1618Retired,

UID.JPEGExtendedHierarchical1719Retired,

UID.JPEGSpectralSelectionHierarchical2022Retired,

UID.JPEGSpectralSelectionHierarchical2123Retired,

UID.JPEGFullProgressionHierarchical2426Retired,

UID.JPEGFullProgressionHierarchical2527Retired,

UID.JPEGLosslessHierarchical28Retired,

UID.JPEGLosslessHierarchical29Retired, UID.JPEGLossless,

UID.JPEGLSLossless, UID.JPEGLSLossyNearLossless,

UID.JPEG2000LosslessOnly, UID.JPEG2000

Class ItemParser.Item		
Retour	**Méthode**	**Explication**
Java.lang;String	**toString()**	Donne un string à l'item

Package org.dcm4che2.imageio.plugins.dcm

DicomImageReadParam	
DicomStreamMetaData	

Class DicomStreamMetaData		
Retour	**Méthode**	**Explication**
org.w3c.dom.Node	**getAsTree(**java.lang.String formatName)	
DicomObject	**getDicomObject()**	Donne un Object
boolean	**isReadOnly()**	
void	**mergeTree(**java.lang.String formatName, org.w3c.dom.Node root)	
void	**reset()**	
void	**setDicomObjec**t(DicomObject dataset)	Enregistre un Object

Class DicomImageReadParam		
Retour	**Méthode**	**Explication**
Java.lang.String	**getOverlayRGB()**	Get the 6 digit hex string that specifies the RGB colour to use for the overlay.
DicomObject	**getPresentationState()**	
Short[]	**getPValue2Gray()**	Obtient les valeurs 2Gray
DicomObject	**getVoiLut()**	Obtient lut de la métadata
Java.lang.string	**getVoiLutFunction()**	Donne le Lut
float	**getWindowCenter()**	@return center
float	**getWindowWidth()**	@return width
boolean	**isAutoWindowing()**	@return vrai si on est en AutoWindowing
void	**setAutoWindowing(**Boolean autoWindowing)	Pour donner un autoWindowing, il faut que le Boolean autoWindowing soit égal à true.
void	**setOverlayRGB**(java.lang;String overlayRGB)	Sets the 6 digit hex string that specifies the RGB colour to use for the overlay.
void	**setPresentationState**(DicomObject prostate)	
void	**setPValue2Gray**(short[] pval2gray)	
void	**setVoiLut**(DicomObject voiLut)	
void	**setVoiLutFunction**(java.lang.String vlutFct)	
void	**setWindowCentre**(float center)	Stocke le centre de la fenêtre

Void	**setWindowWidth**(float width)	Stocke la largeur de la fenêtre

Package org.dcm4che2.imageioimpl.plugins.dcm

DicomImageReader	An abstract superclass for parsing and decoding of images.
DicomImageReaderSpi	Constructs a blank DicomImageReaderSpi.
DicomImageWriter	Write DICOM files containing images - handles transcoding to a new representation, as well as transcoding the images.
DicomImageWriterSpi	Provide information about the dicom image writers

Class DicomImageReader		
boolean	**canReadRaster()**	Returns true if this plug-in supports reading just a Raster of pixel data.
void	**copyReadParam(**javax.imageio.ImageReadParam src, javax.imageio.ImageReadParam dst)	
javax.imageio.ImageTypeSpecifier	**createImageSpecifier()**	Create an image type specifier for the entire image
void	**dispose()**	Allows any resources held by this object to be released.
jvax.imageio.ImageReadParam	**getDefaultReadParam()**	
int	**getHeigh**t(int imageIndex)	Returns the height in pixels of the given image within the input source.
javax.imageio.metadata.IIOMetadata	**getImageMetadata**(int imageIndex)	Gets any image specific meta data.
java.util.Iterator<javax.imageio.ImageTypeSpecifier>	**getImageTypes(**int imageIndex)	Returns an Iterator containing possible image types to which the given image may be decoded, in the form of ImageTypeSpecifierss (dicom).
int	**getNumImage**(boolean allowSearch)	Returns the number of regular images in the study.
javax.imageio.metadata.IIOMetadata	**getStreamMetadata()**	Return a DicomStreamMetaData object that includes the DICOM header.

Class DicomImageReader(suite)		
Retour	**Méthode**	**Explication**
int	**getWidth(**int imageIndex)	Returns the width in pixels of the given image within the input source.
void	**initImageReader(**int imageIndex)	Sets the input for the image reader.
void	**postDecompress()**	
Java.awt.image.BufferedImage	**read(**int imageIndex, javax.imageio.ImageReadParam param)	Reads the provided image as a buffered image.
byte[]	**readBytes(**int imageIndex, javax.imageio.ImageReadParam param)	Reads the bytes for the given image as raw data.
Java.awt.image.Raster	**readRaster(**int imageIndex, javax.imageio.ImageReadParam param)	Read the raw raster data from the image, without any LUTs being applied.
void	**reset()**	Restores the ImageReader to its initial state.
void	**setInput**(java.lang.Object input, boolean seekForwardOnly, boolean ignoreMetadata)	Sets the input source to use to the given ImageInputStream or other Object.
int	**getNumImages(**boolean allowSearch)	Returns the number of regular images in the study.

Class DicomImageReaderSpi		
Retour	**Méthode**	**Explication**
boolean	**canDecodeInput(**java.lang.Object input)	Returns true if the supplied source object appears to be of the format supported by this reader.

javax.imageio.ImageReader	**createReaderInstance(**java.lang.Object extension)	Returns an instance of the ImageReader implementation associated with this service provider.
java.lang.String	**getDescription(**java.util.Locale locale)	Returns a brief, human-readable description of this service provider and its associated implementation.

Class DicomImageReaderSpi(suite)		
Retour	**Méthode**	**Explication**
boolean	**canWriteSequence()**	Writing as a sequence is actually the preferred approach.
javax.imageio.metadata.IIOMetadata	**convertImageMetadata(** javax.imageio.metadata.IIOMetadata metadata, javax.imageio.ImageTypeSpecifier type, javax.imageio.ImageWriteParam param)	The image metadata provided MUST be of the correct type for the child image writer, as the meta-data will only be written to the child image.
javax.imageio.metadata.IIOMetadata	**converStreamMetadata(** javax.imageio.metadata.IIOMetadata metadata, javax.imageio.ImageWriteParam param)	Can't convert anything except existing DicomStreamMetaData
void	**endWriteSequence()**	Finish writing the header data to the stream.
byte[]	**extracyImageEncoding(** javax.imageio.IIOImage iioimage, javax.imageio.ImageWriteParam param)	This method gets the encoded image from the given object as a byte array of data.
javax.imageio.metadata.IIOMetadata	**getDefaultImageMetadata**(javax.imageio.ImageTypeSpecifier image, javax.imageio.ImageWriteParam writeParam)	No easy way to figure out what htis one as as the stream meta-data isn't yet available.
javax.imageio.metadata.IIOMetadata	**getDefaultStreamMetadata** (javax.imageio.ImageWriteParam arg0)	Get a default set of DICOM data to use in the stream meta-data.
void	**prepareWriteSequence**(javax.imageio.metadata.IIOMetadata metadata)	Start writing the intiial sequence.

void	**setupWriter**(javax.imageio.metadata.IIOMetadata metadata)	Sets up the child writer if it hasn't already been setup
void	updateDicomHeader((DicomStreamMetaData metadata, java.awt.image.BufferedImage image)	This updates the metadata with relevant information from the image, such as the size etc.
void	**write**(avax.imageio.metadata.IIOMetadata metadata, javax.imageio.IIOImage iioimage, javax.imageio.ImageWriteParam param)	Writes the image, including the DICOM metadata.
void	**writeBytesToSequence(** byte[] data, javax.imageio.ImageWriteParam param)	Write the given image as a byte array to the sequence.
void	**writeToSequence(** javax.imageio.IIOImage iioimage, javax.imageio.ImageWriteParam param)	Write the given image to the sequence.

Class DicomImageWriterSpi		
Retour	**Méthode**	**Explication**
boolean	**canEncodeImage**(javax.imageio.ImageTypeSpecifier type)	Indicate the DICOM can encapsulate the given type
javax.imageio.ImageWriter	**createWriterInstance(** java.lang.Object extension)	Create a dicom image writer
Java.lang.Strng	**getDescription(**java.util.Locale locale)	Get the description of this image writer type.

Package org.dcm4che2.imageioimpl.plugins.rle

RLEImageReader	
RLEImageReaderSpi	

Class RLEImageReader		
Retour	**Méthode**	**Explication**
int	**getHeight**(int imageIndex)	Returns the height in pixels of the given image within the input source.
javax.imageio.metadata.IIOMetadata	**getImageMetaData(**int imageIndex)	Returns an IIOMetadata object containing metadata associated with the given image, or null if the reader does not support reading metadata, is set to ignore metadata, or if no metadata is available.
java.util.Iterator<javax.imageio.ImageTypeSpecifier>	**getImageTypes(**int imageIndex)	Returns an Iterator containing possible image types to which the given image may be decoded, in the form of ImageTypeSpecifierss.
int	**getNumImages**(boolean allowSearch)	Returns the number of images, not including thumbnails, available from the current input source.
javax.imageio.metadata.IIOMetadata	**getStreamMetadata**(int imageIndex)	Returns an IIOMetadata object representing the metadata associated with the input source as a whole (i.e., not associated with any particular image), or null if the reader does not support reading metadata, is set to ignore metadata, or if no metadata is available.
Int	**getWidth**(int imageIndex)	Returns the width in pixels of the given image within the input source.
java.awt.image.BufferedImage	**read**(int imageIndex, javax.imageio.ImageReadParam param)	Reads the image indexed by imageIndex and returns it as a complete BufferedImage, using a supplied ImageReadParam.
void	**setInput**(java.lang.Object input, boolean seekForwardOnly, boolean ignoreMetadata)	Sets the input source to use to the given ImageInputStream or other Object.

Class RLEImageReaderSpi		
Retour	**Méthode**	**Explication**
boolean	**canEncodeImage**(javax.imageio.Image TypeSpecifier type)	Indicate the DICOM can encapsulate the given type
javax.imageio.Imag eWriter	**createWriterInstance(** java.lang.Object extension)	Create a dicom image writer
Java.lang.Strng	**getDescription(**java.util.Locale locale)	Get the description of this image writer type.

Package org.dcm4che2.io

ContentHandlerAdapter	Receive notification of the logical content of a document.
DataOutputStreamAdapter	This class adapts a regular output stream onto a data output stream, such as would be found in an ImageOutputStream
DicomInputStream	
DicomOutputStream	
ImageInputStreamAdapter	
SAXReader	
SAXWriter	
StopTagInputHandler	
TranscoderInputHandler	

Class ContentHandlerAdapter		
Retour	**Méthode**	**Explication**
void	**characters(**char[] ch, int start, int length)	Receive notification of character data dicom.
void	**endElement(** java.lang.String namespaceURI, java.lang.String localName, java.lang.String qName)	Receive notification of the end of an element
void	**setDocumentLocator(o**rg.xml.sax.Locator locator)	Receive an object for locating the origin of SAX document events.
Class DataOutputStreamAdapter		
Retour	org.xml.sax.Attributes atts) **Méthode**	**Explication**

void	**writes**(byte[] b, int off, int len)	Writes len bytes from array b, in order, to the output stream.
void	**write (**int b)	Writes to the output stream the eight low-order bits of the argument b.

Class DicomInputStream		
Retour	**Méthode**	**Explication**
DicomObject	**`getDicomObject`**()	Obtient un Object du dicom donc lecture des métadata
long	**`getEndOfFileMetaInfoPosition`**()	Obtient la fin des métadatas du dicom
byte[]	**`getPreamble`**()	
long	**`getStreamPosition`**()	
TransferSyntax	**`getTransferSyntax`**()	
int	**level()**	
void	**mark**(int readlimit)	Marks the current position in this input stream.
int	**read**(byte[] b, int off, int len)	Reads up to len bytes of data from this input stream into an array of bytes. This method blocks until some input is available.
DicomObject	**`readDicomObject`**()	Lecture des métadonnées dicom
void	**readDicomObject**(DicomObject dest, int len)	
DicomObject	**`readFileMetaInformation`**()	Read File Meta Information from this stream. If there is no File Meta Information on current stream position, null is returned. Otherwise the stream will be parsed until the

		end of the File Meta Information is detected and a DicomObject containing the File Meta Information is returned.
void	**readFully**(byte[] b)	
void	**readFully**(byte[] b, int off, int len)	
int	**readHeader**()	Lecture de l'entête.
void	**readItem**(DicomObject dest)	Lecture des Items
vodi	**readItems**(DicomElement sq, int sqlen)	
boolean	**readValue**(DicomInputStream dis)	Called by the input stream when reading a DICOM value in the stream.
void	**reset**()	Repositions this stream to the position at the time the mark method was last called on this input stream.
void	**setEndOfFileMetaInfoPosition**(long fmi EndPos)	
void	**setHandler**(DicomInputHandler handler)	
void	**setStreamPosition**(long pos)	
long	**skip**(long n)	
DicomElement	**sq**()	
int	**tag**()	
long	**tagPosition**()	

int	**valueLength**()	
VR	**vr**()	

Class DicomOutputStream		
Retour	**Méthode**	**Explication**
void	**close**()	Closes this file output stream and releases any system resources associated with this stream.
void	**finish**()	Finishes writing compressed data to the output stream without closing the underlying stream. Use this method when applying multiple filters, and the transfer syntax is a deflator transfer syntax.
byte[]	**getPreamble**()	.
long	**getStreamPosition**()	
TransferSyntax	**getTransferSyntax**()	
boolean	**isAutoFinish**()	Indicate if the stream is finished automatically (compressed data written) when the dataset is written
boolean	**isExplicitItemLength**()	
boolean	**isExplicitItemLengthIfZero**	
boolean	**isExplicitSequenceLength**()	

Class DicomOutputStream(suite)		
Retour	**Méthode**	**Explication**
boolean	**isExplicitSequenceLengthIfZero**()	.
void	**serializeDicomObject**(DicomObject attrs)	Only for internal use by DicomObjectSerializer.
void	**setAutoFinish**(boolean autoFinish)	.Set to false to not auto finish the stream on writing a data set - useful for writing a DataObject followed by some additonal DICOM that is custom written, eg images or related large data.
void	**setExplicitItemLength**(boolean explicitItemLength)	
void	**setExplicitItemLengthIfZero(**boolean explicitItemLengthIfZero)	
void	**setIncludeGroupLength**(boolean includeGroupLength	Indicate if the stream is finished automatically (compressed data written) when the dataset is written
void	**setPreamble**(byte[] preamble)	
void	**setStreamPosition**(long pos)	
void	**setTransferSyntax**(java.lang.String tsuid)	
void	**write**(byte[] b, int off, int len)	Writes len bytes from the specified byte array starting at offset off to this file output stream.
void	**write**(int b)	Writes the specified byte to this output stream
void	**writeCommand**(DicomObject attrs)	
void	**writeDataset**(DicomObject attrs, java.lang.String tsuid)	Write a DICOM dataset to the output stream.

void	**writeDataset**(DicomObject attrs, TransferSyntax transferSyntax)	Write a DICOM dataset to the output stream.
void	**writeDicomFile(**DicomObject attrs)	Write a DICOM object to the output stream using the specified DicomObject to obtain the transfer syntax UID and other attributes.
void	**writeFileMetaInformation**(DicomObject attrs)	
void	**writeHeader**(int tag, VR vr, int len)	
void	**writeItem**(DicomObject item, TransferSyntax transferSyntax)	Write an item (DicomObject) to the output stream.

Class ImageInputStream		
Retour	**Méthode**	**Explication**
void	**mark**(int readlimit)	Marks the current position in this input stream.
boolean	**markSupported**()	Tests if this input stream supports the mark and reset methods.
int	**read**()	Reads the next byte of data from the input stream.
int	**read**(byte[] b,int off, int len)	Reads up to len bytes of data from the input stream into an array of bytes.
void	**reset()**	Repositions this stream to the position at the time the mark method was last called on this input stream.
long	**skip(long n)**	Skips over and discards n bytes of data from this input stream.

Class SAXReader		
Retour	**Méthode**	**Explication**
DicomObject	**readDicomObject()**	Lit les métadatas du dicom
DicomObject	r**eadDicomObject**(DicomObject dcmObj)	

Class SAXWriter		
Retour	**Méthode**	**Explication**
`java.io.File`	**`getBaseDir`**`()`	
Int[]	**`getExclude`**`()`	
boolean	**`readValue`**`(DicomInputStream in)`	Called by the input stream when reading a DICOM value in the stream.
void	**setBaseDir**(java.io.File baseDir)	
void	**`setExclude`**`(int[] exclude)`	
void	**`write`**`(DicomObject attrs)`	

Class StopTagInputHandler		
Retour	**Méthode**	**Explication**
boolean	**`readValue`**`(DicomInputStream in)`	Called by the input stream when reading a DICOM value in the stream.

Class TranscoderInputHandler		
Retour	**Méthode**	**Explication**
boolean	**readValue**(DicomInputStream in)	Called by the input stream when reading a DICOM value in the stream.

Package org.dcm4che2.data

This package contains lower level classes that deal with the DICOM Data Dictionary (PS3.6) Data Structure and Encoding (PS3.5).

Please note that some of these classes (such as Tag) are automatically generated by XSLT from DICOM Standard Text MS Word documents. So your amendments and/or modifications will be lost with the next run of dcm4che2/dcm4che2-core/dcm4che2-core-dict/build.xml to include attributes defined in final text supplements and correction items additional to Part 6 and 7 of the base standard text.

BasicDicomObject	
DateRange	
DicomObjects	
DicomObjectToStringParam	
ElementDictionary	
Implementation	
PersonName	
RessourceLocator	
SpecificCharacterSet	
Tag	Provides tag constants
TransferSyntax	
UID	Provides tag constants
UIDDictionary	
VR	This class provides enumeration of DICOM Value Representation
VRMap	

Class BasicDicomObject		
Retour	**Méthode**	**Explication**
boolean	**accept**(DicomObject.Visitor visitor)	Calls @link{Visitor#visit} for each element in this Dataset. Returns false, if @link{Visitor#visit} returns false for any element. **Parameters:** visitor - *Visitor* object, which method @link{Visitor#visit} is called for each element in this Dataset.
void	**add(**DicomElement a)	
boolean	**bigEndian()**	Called by the input stream when reading a DICOM value in the stream.
void	**bigEndian**(boolean bigEndian)	
boolean	**cacheGet()**	
void	**cacheGet(**boolean cacheGet)	
void	**clear()**	Removes all elements from this Dicom Object.
DicomObject	**command()**	
Java.util.Iterator<DicomElement>	**commandIterator()**	Returns an iterator over Command elements (0000, eeee) in this Dicom Object.
boolean	**contains(**int tag)	Returns true, if this Dicom Object contains the specified Element.
boolean	**containsAll(**DicomObject keys)	Returns true if this Dicom Object contains all of the elements in the specifield Dicom Object.
boolean	**containsValue(**int tag)	Returns true, if this Dicom Object contains the specified Element with a value length > 0.
void	**copyTo(**DicomObject dest)	

DicomObject	**dataset()**	
Java.util.Iterator<DicomElement>	**datasetIterator()**	Returns an iterator over Data elements (group tag > 2) in this Dicom Object.
boolean	**equals**(java.lang.Object o)	

Class BasicDicomObject(suite)		
Retour	**Méthode**	**Explication**
DicomObject	**exclude(**int[] tags)	
DicomObject	**excludePrivate()**	
DicomObject	**fileMetaInfo()**	
Java.util.Iterator<DicomElement>	**fileMetaInfoIterator()**	Returns an iterator over File Meta Information elements (0002, eeee) in this Dicom Object.
DicomElement	**get(**int tag)	
DicomElement	**get(**int[] tagPath**)**	
DicomElement	**get(**int[] tagPath, VR vr**)**	
byte[]	**getBytes(**int tag)	
byte[]	**getBytes(**int[] tagPath)	
byte[]	**getBytes(**int[] tagPath, boolean bigEndian)	
byte[]	**getBytes(**int tag, boolean bigEndian)	
java.util.Date	**getDate**(int tag)	
java.util.Date	**getDate**(int[] tagPath)	
java.util.Date	**getDate**(int[] tagPath, java.util.Date defVal)	

java.util.Date	**getDate**(int[] itemPath, int daTag, int tmTag)	
java.util.Date	**getDate**(int[] itemPath, int daTag, int tmTag, java.util.Data defVal)	
java.util.Date	**getDate**(int tag,VR vr)	
java.util.Date	**getDate**(int tag,VR vr, java.util.Date defVal)	
DateRange	**getDateRange(**int tag)	
DateRange	**getDateRange(**int[] tagPath)	
DateRange	**getDateRange(**int[] tagPath, DateRange defVal)	
DateRange	**getDateRange(**int[] tagPath, int daTag, int tmTag)	
DateRange	**getDateRange(**int[] tagPath, int daTag, int tmTag, DateRange defVal)	
DateRange	**getDateRange(**int[] tagPath, VR vr)	
DateRange	**getDateRange(**int[] tagPath, VR vr, DateRange defVal)	

Class BasicDicomObject(suite)		
Retour	**Méthode**	**Explication**
DateRange	**getDateRange(**int tag, DateRange defVal)	
DateRange	**getDateRange(**int daTag, int tmTag)	
DateRange	**getDateRange(**int daTag, int tmTag, DateRange defVal)	
java.util.date[]	**getDates(**int tag)	
java.util.date[]	**getDates**(int[] tagPath, java.util.Date[] defVal)	
java.util.date[]	**getDates**(int[] tagPath, VR vr, java.util.Date[] defVal)	
java.util.date[]	**get(**int[] tagPath, VR vr**)**	

III) Différent principe pour dicom:

Les méthodes décrient son selon le package dcm4che2.

III.1) Gestion de la liste des tags en DICOM pour un fichier standard (une seule image dans le fichier) :

a) But :

On souhaite afficher une image.

b) Entrée en java Standard :

Les bibliothèques d'E/S utilisent souvent l'abstraction d'un flux [stream], qui représente n'importe quelle source ou réceptacle de données comme un objet capable de produire et de recevoir des parties de données. Le flux cache les détails de ce qui arrive aux données dans le véritable dispositif d'E/S.

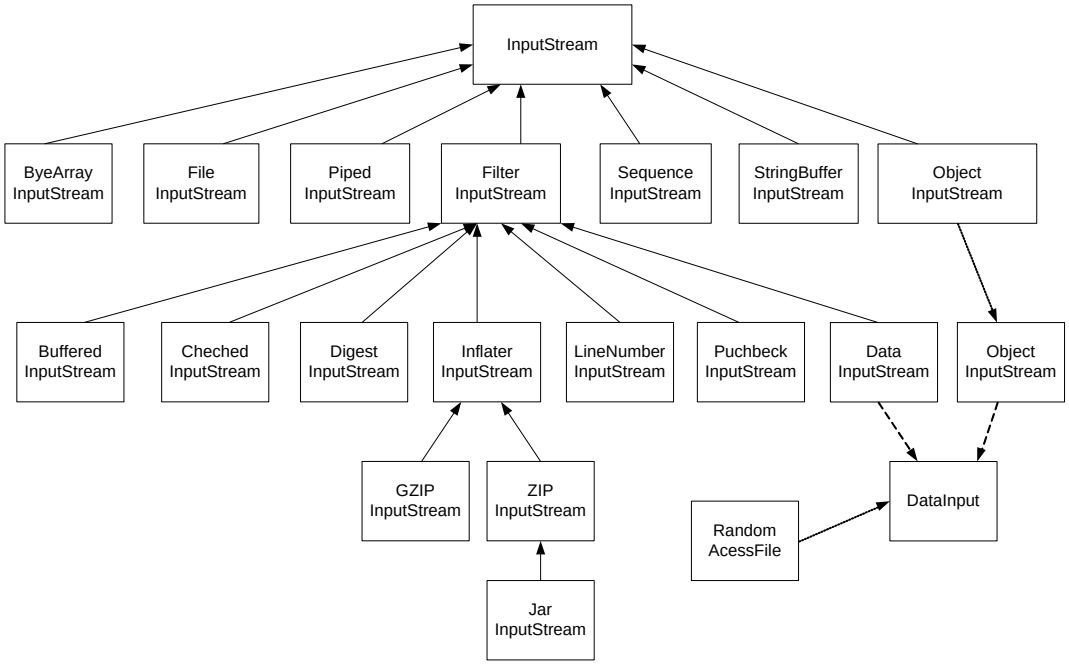


Figure 1 : schéma de l'entrée

Sous-classe	Fonction
FileInputStream	Permet de créer un flux avec un fichier présent dans le système de fichiers. Cette classe possède un constructeur prenant en paramètre un objet de type File ou un String , qui représente le chemin vers le fichier.
ByteArrayInputStream	Permet de lire des données binaires à partir d'un tableau d'octets.
PipedInputStream	Permet de créer une sorte de tube d'entrée (pipe). Dans celui-ci, les informations circuleront sous forme d'octets. Cette classe possède un constructeur ayant pour paramètre un objet de type PipedOutputStream. On peut ainsi connecter les deux tubes ; en gros, ce qui est écrit dans une extrémité peut être lu depuis l'autre.
BufferedInputStream	Cette classe permet la lecture de données à l'aide d'un tampon, un buffer si vous préférez. À l'instanciation, un tableau d'octets est créé afin de servir de tampon et permet de ne pas surcharger la mémoire. Ce tableau est redimensionné automatiquement à chaque lecture pour contenir les données provenant du flux d'entrée. Ce type d'objet est particulièrement approprié lors de traitement de fichiers volumineux !
DataInputStream	Cet objet sert à lire des données représentant des types primitifs de Java (int, boolean, double, byte, ...) préalablement écrits par un DataOutputStream. Grâce à cet objet, vous pouvez récupérer des éléments sérialisés avec des méthodes comme readInt(), readDouble()...
PushbackInputStream	Lit un flux binaire en entrée et replace le dernier octet lu dans le flux !
LineNumberInputStream	Permet d'avoir les numéros de lignes lues dans le flux en plus de lire le flux lui-même.
SequenceInputStream	Permet de concaténer deux (ou plus) flux d'entrée, ce qui permet de traiter plusieurs flux d'entrée comme un seul et unique flux !
ObjectInputStream	Permet de «désérialiser» un objet, c'est-à-dire de restaurer un objet préalablement sauvegardé à l'aide d'un ObjectOutputStream. Cet objet est l'homologue de l'objet DataInputStream, à la différence que celui-ci traite des objets.

Tableau 1 : Rappel de la gestion d'ouverture d'un fichier en java

c) Lire/Écrire dans des fichiers

Le principe d'ouverture d'un fichier est très simple. On fait une lecture d'octet de donnée donc un flux d'information.

FileInputStream : dérivée d'*InputStream* permet d'effectuer une lecture d'octets dans un fichier.

On associe à un objet de type *File,* le flux correspond alors à un flux de fichier :

```
File f =new File(« nomDuFichier »)

FileInputStream f1 = new FileInputStream(f) ;
```

La lecture d'un octet se fait alors avec la méthode *read* :

```
Byte b = f1.read()
```

d) *Lire/Écrire des données numériques :*

La classe *DataInputStream* permet de lire des données de type numériques tels que les flottants, les booléens, les octets ou les entiers (type double, booléen, byte et int).

Méthodes de *DataInputStream* (Non exhaustif)	
int	`read(byte[] b)` Lit l'ensemble des octets du flux et les stocke dans un tableau. Renvoie –1 si la fin du flux a été atteinte.
int	`read(byte[] b, int off, int len)` Lit le flux à partir de *off* sur *len* octets. Renvoie –1 si la fin du flux est atteinte
boolean	`readBoolean()` Renvoie un booléen égal à *true* si l'octet lu est non nul et *false* si l'octet est nul
byte	`readByte()` Renvoie l'octet lu.
char	`readChar()` Renvoie le caractère correspondant à l'octet lu
double	`readDouble()` Lit 8 octets et renvoie le double correspondant.

e) Chainage des flux filtrés :

```
FileInputStream fis = new FileInputStream(“nomFichier” );

DataInputStream dis = new DataInputStream(fis);

Double s = dis.readDouble();
```

Cet enchainement va vous permettre d’associer les différents types de flux de manière à bénéficier de leurs fonctionnalités. En effet, partons du principe que *DataInputStream* sait lire des numériques et *FileInputStream* sait lire dans des fichiers et associons les pour qu’ils nous fassent partager leurs compétences propres.

```
FileInputStream dis = new FileInputStream(new BufferedInputStream
        (new FileInputStream(“fiche.dat”)));
```

L’objet de type DataInputStream est haut de la chaîne de flux puisque ses méthodes d’accès aux données nous serons utiles pour lire des données de type numériques. La méthode read() de l’objet « dis » va procéder à une lecture bufférisée dans le flux.

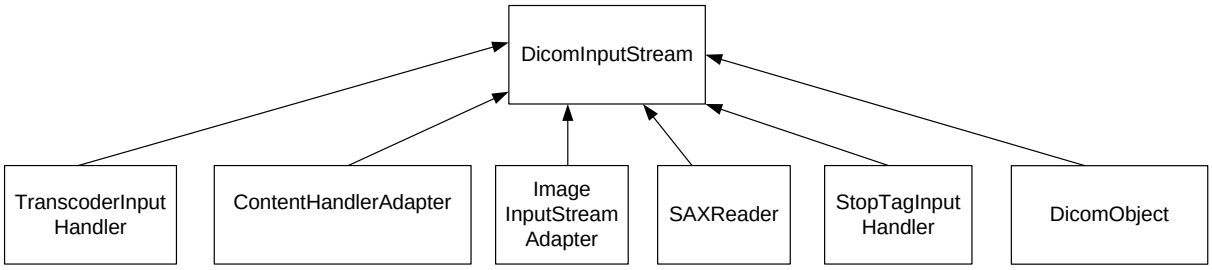
f) Lecture dans un fichier

Cet exemple d’ouverture d’un fichier .txt.

```
String fileString = “”;

FileReader fr;

try {

        fr = new FileReader(“data.txt”);

        int i=0;

        while ((I = fr.read()) !=-1;

        fr.close();
```

g) Lecture dans un fichier DICOM:

On va se rappeler par un petit graphe les différentes structures de ce package pour l'ouverture d'un fichier.



Pour ouvrir une image en DICOM, nous avons cette suite instruction suivante :

- Filtrer le fichier entrée en vérifiant que c'est un fichier DICOM : avec DicomInputStream(File file)
- On lit les tags dans le fichier donc les métadonnées DICOM avec DicomObject

```
DicomInputStream dis = new DicomInputStream(new File("im2.dcm"));

DicomObject object = dis.readDicomObject();
```

Mais pour trouver les données DICOM rangé dans une image, on doit parcourir chaque ligne du fichier.

Pour comprendre le phénomène de lecture des métadonnées, nous devons faire un petit rappel sur le standard DICOM.

RAPPEL :

Un fichier DICOM est composé d'un fichier XML qui est composé d'une étiquette (Tag) de la matrice de pixels de l'image + d'autres métadonnées.

Les métadonnées sont rangées comme une file d'attente. Les tags sont toujours rangés de la même façon.

Le standard DICOM est basé sur un encodage binaire de ces métadonnées sur les normes ISO actuelles de l'imagerie (JPEG, JPEG2000). On dit métadonnées les informations qui donnent des renseignements de l'image.

Nous allons dans un premier temps voir la structure de l'encodage binaire des métadonnées du standard DICOM. On prend l'exemple classique de David CLUNIE (fondateur de DICOM 3) :

Nous souhaitons par exemple trouver la colonne (donc la hauteur de l'image DICOM)

On définit **TAG** (étiquette) **(fig2)** est une paire ordonnée de 16 bits d'entiers non-signés représentant le numéro du groupe suivi par le numéro de l'élément. Le tag est représenté par l'attribut « **group** » qui représente un groupe d'élément de donnée caractéristique. On peut faire analogie avec le langage JAVA c'est la classe par exemple la class patient.

Puis le deuxième élément « **element** » est son les caractéristiques physiques, donnée spécifique de la classe. Reprenons notre exemple de class patient, on pourrait avoir le nom du patient, son âge, son adresse…

Le **VR** signifie la Value Representation. Le VR est une chaîne de deux caractères qui est codée en 2 octets associée à une certaine valeur du Data Tag Element. Chaque TAG doit avoir un VR qui est fixé par le standard DICOM. Les VR sont définies dans le standard DICOM dans la partie 6 nommée le Data Dictionary.

Le **Length** est défini suivant deux ensembles :

- soit un entier non-signé de 16 ou 32 bits dépendant d'un VR explicite ou implicite, et contenant la longueur explicite du champ de valeur en nombre d'octets.
- soit un ensemble de champs de longueur 32 bits. Les longueurs indéfinies pourraient être utilisées pour les éléments de données ayant le VR égal à SQ (Sequence of items) et UN (Unknown ou autres)

Le **Value Field** est un champ de valeur et de longueur variable. Il correspond à l'information identifié par le premier champ. Le type de données de valeurs enregistré dans ce champ est spécifié par le VR de l'élément de données. Par exemple si VR a pour normalisation CS (Code String) alors dans le champ value nous devons avoir une chaîne de caractère, si VR est IS (Integer String) alors dans le champ nous aurons un entier (vue par l'utilisateur). Le champ value est l'image des caractères ASCII ce qui signifie que value est une valeur codée en binaire.

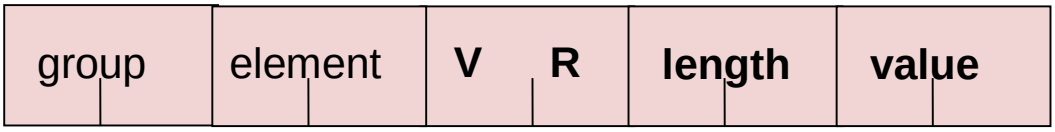
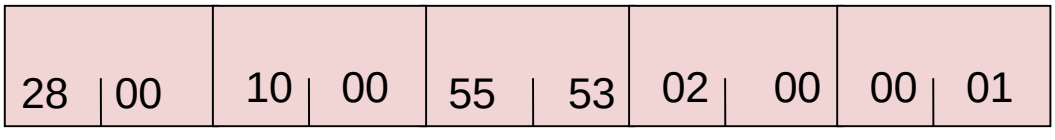


Figure 2: définit le fonctionnement d'une ligne métadonnée

Par ce dernier point, on vient de voir que la spécificité du standard DICOM n'est pas facile par rapport à la représentation actuelle XML, PHP. Alors ce type de codage pour Dicom n'est pas facile avec l'encryptage en binaire.

Prenons comme exemple pour voir le codage binaire :

Tag (0028, 0010) qui représente le nombre de lignes de l'image. Le champ de longueur est représenté en binaire non-signé (Unsigned Short) sur deux octets. On aura par exemple une valeur totale de ligne dans cette image de 256.



Pour notre exemple (0028,0010) à pour VR = US qui est égale à une longueur de 2 octets (2*8bits).

On peut résumer notre analyse par cette figure qui représente le nombre de bits.

On peut voir que le tag est représenté par le regroupement du group et element qui sont chacun codés sur 16 bits (2 octets). Puis de la VR(Value Representation) qui est de 0, ou 16 ou 32 bits.

On poursuit cette séquence par la longueur de la « value » qui peut prendre soit 16 ou 32 bits.

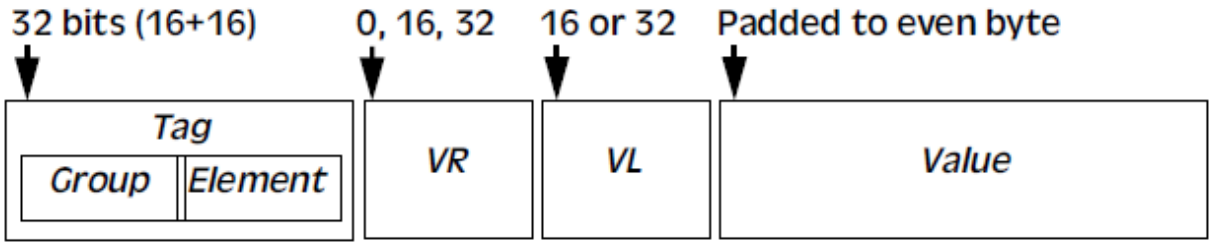
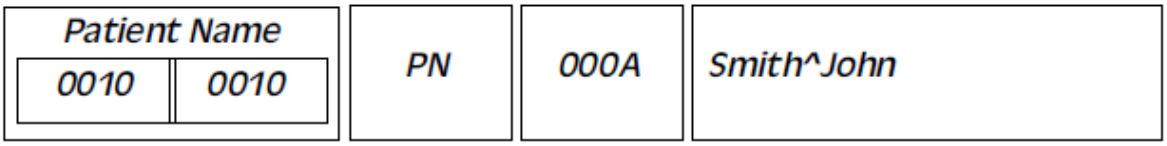


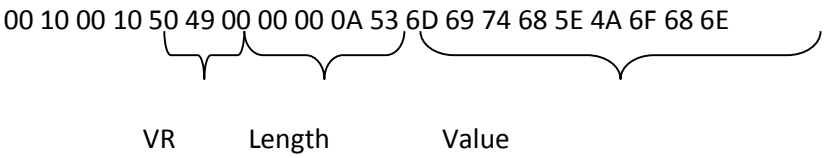
Figure 3: représente nombre bits selon le champ voulu.

Autre exemple pour bien comprendre :



On a tag (0010, 0010) qui signifie que s'est le nom du patient, un VR = PN(Person Name) qui a une longueur de 64 caractères par groupe.

Si on traduit cette ligne comme la machine comprend donc en code hexadécimale[1] :



L'explication de VL est la suivante extraite du livre David Clunie anglais :

The number of bits used to encode the VR and VL fields depends on the transfer syntax and the VR. In the default implicit VR little endian transfer syntax, the VR is never sent and the VL is always 32 bits. In the all other transfer syntaxes, the VR is always sent and is 16 bits. For OB, OW, SQ, UT and UN VRs, the 16 bit VR is followed by 16 bits of zeroes, then a 32 bit VL. For all other VRs, a 16 bit VR field is followed by a 16 bit VL. The idea is to stay aligned on a 32 bit boundary when 32 bit VLs are required for "big" values, and save 32 bits for the others. In retrospect, this "optimization" has caused nothing but trouble, particularly when adding new VRs to the standard, or trying to send "big" values in "small" VRs.

On va voir maintenant : comment les métadonnées sont rangés dans un fichier de type DICOM. On sait que les métadonnées sont codées par-dessus un standard d'image classique comme JPEG-2000.

[1] Voir annexe : code ASCII

Les éléments de données sont ensuite ordonnés par ordre croissant pour former un dataset, corps du message. On peut dire en gros que s'est une encapsulation des entêtes comme une file d'attente.

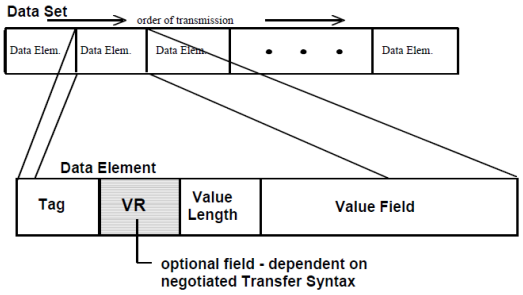


figure 3 : explication la façon d'encodage des métadonnées

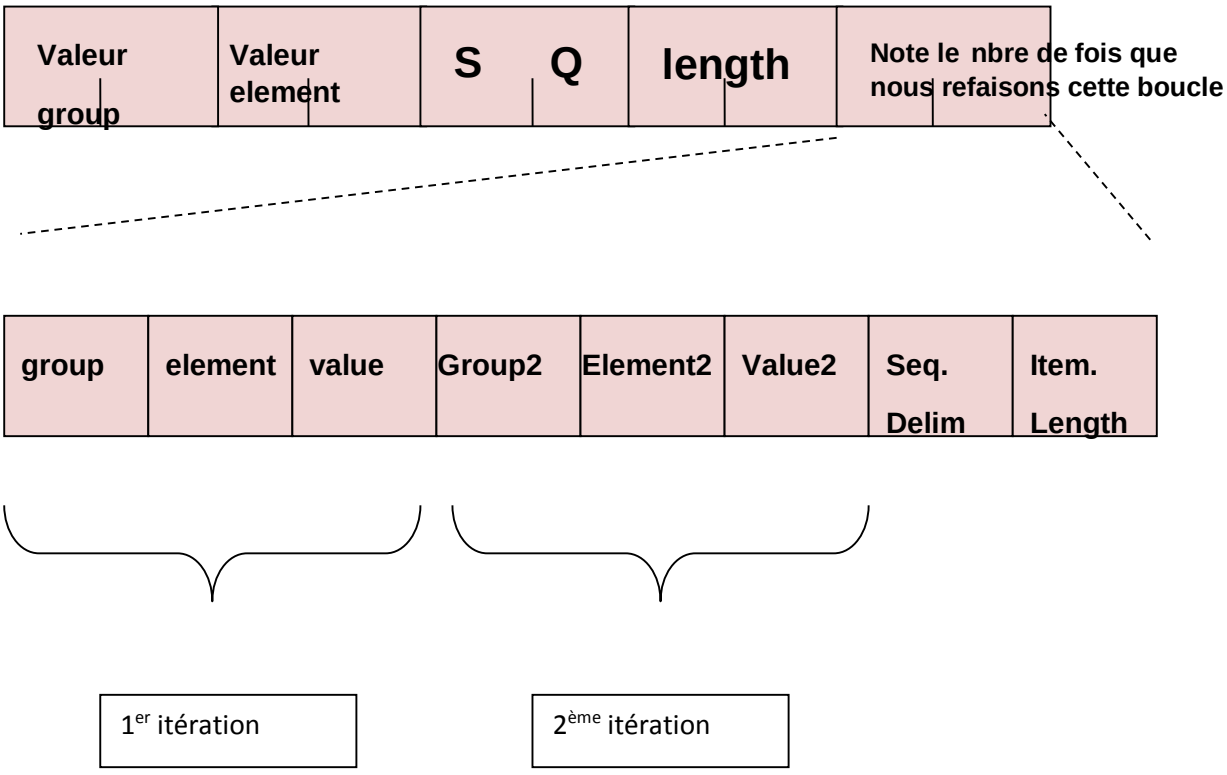
Il existe un cas spéciale dans l'encodage de DICOM quand nous avons l'attribut VR = SQ signifie Sequence of Element. Ce VR représente « Valeur de séquence de zéro ou plusieurs séquences ».

(a) Représente la façon d'être ranger dans les métadonnées :

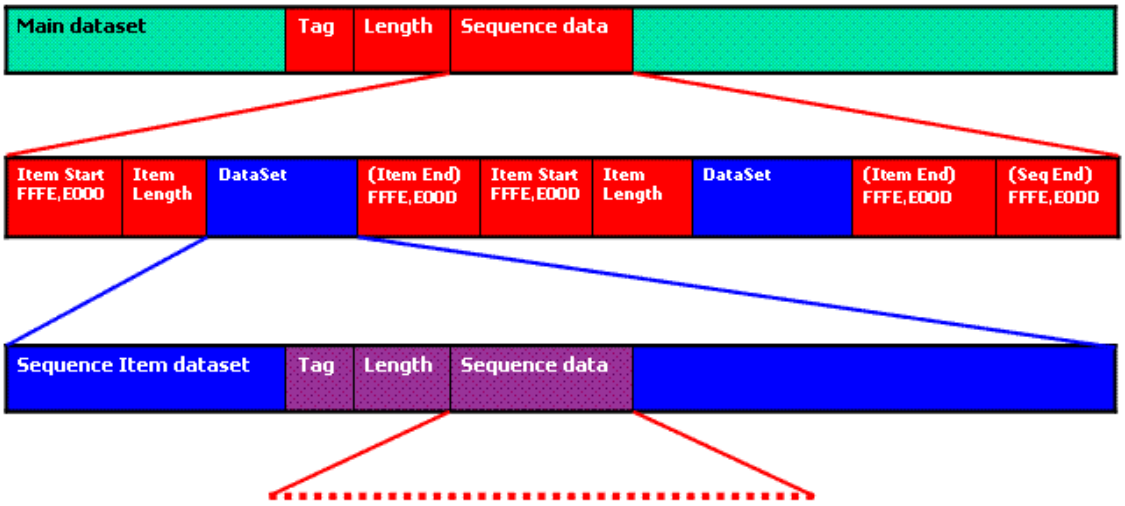
On reprend notre schéma que nous avons vu sur l'encodage d'une seule métadonnée.

group	element	V R	length	value

Nous prenons ce cas spécifique que VR = SQ :



(b) Représentation général de ce phénomène :



On a parlé jusqu'à maintenant de l'encodage mais nous avons dit que les métadonnées sont rangées comme une file d'attente. Il a dû faire un dictionnaire des différents tag classée selon un certain autre pour permettre d'avoir la bonne grammaire (permettre au programme de décodée la bonne valeur lu en binaire).

Tout fichier DICOM doit avoir cette structure générale pour l'ensemble des métadonnées suivante :

Ce standard est un modèle très structurés car nous avons toujours cet ordre : Patient, Etude, Série, Image.

Nous avons des UID (Unique IDentifier) pour l'étude, une série, une image.

Les UID permettent dire que l'image est unique par toutes images faites.

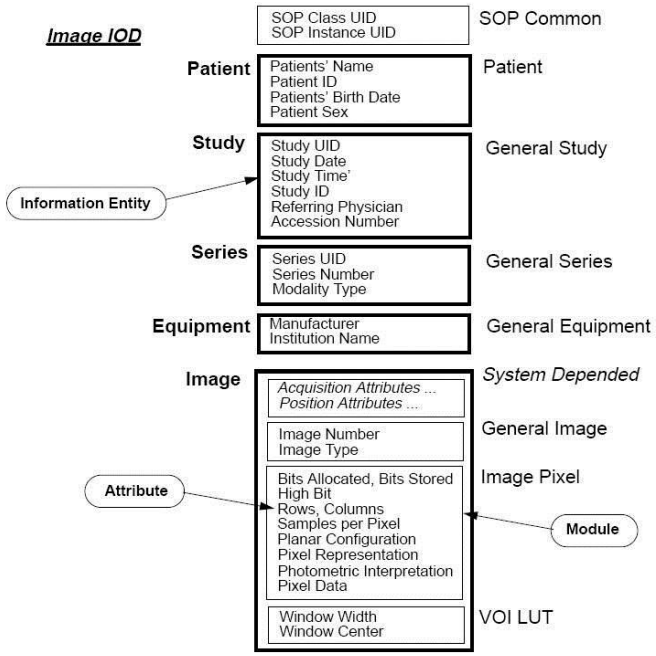


Figure 4 : schéma du classement des métadonnées

Le SOP Class sont des données qui renseignent sur la communication machine et serveur de stockage. Ils sont donnés automatiquement par la machine d'imagerie.

On a vu un petit rappel de la structure des tags. Nous aurons toujours une image dicom sous la forme suivante :

Element Tag and Value		Binary Coding
0008,0000	**726**	**08 00 00 04 00 00 00 D6 02 00 00 ($726_2 = 2D4_{16}$)**
0008,0005	**ISO.IR 100**	**08 00 05 00 0A 00 00 00 49 53 4F 5F 49 52 20 31 30 30**
0008,0016	**1.2.840.10008.5.1.4.1.1.2**	**08 00 16 00 1A 00 00 00 31 2E 32 2E 38 34 30 2E 31 30 30 30 38 2E 35 2E 31 2E 34 2E 31 2E 31 2E 32 00**
0008,0060	**CT**	**08 00 60 00 02 00 00 00 43 54**

Explication de cet exemple :

« 0008,0000 » dans le « Element Tag and Value » column est le tag pour 0008 Group « 726 » est la valeur pour le Group length et signifie qu'il y a 726 octets dans son Group. La correspondance code binaire de son tag et la valeur sont codés en lignes « Binary coding ». Les prochaines lignes sont les tags et values peut correspondre « Specific Character Set », « SOP Class UID », « Modality », et « Study Description »

Pour lire l'ensemble des métadonnées DICOM, dcm4che2 utilise le principe itération :

Le principe d'itération en java est le suivant : (iterator)

Dans java native :

Une des méthodes de l'interface **Collection**, à savoir **Iterator iterator()** vise à construire un itérateur pour une collection. Un itérateur (instance d'une classe qui implante l'interface Iterator<E>) permet d'énumérer les éléments contenus dans une collection.

Iterator est une interface dont **ListIterator** est une sous-interface adaptée aux listes.

Au travers d'une implantation de cette interface, il doit être possible de parcourir les objets constituant une collection.

Les trois opérations réalisées au cours d'un tel parcours sont :

- une initialisation,
- un test de fin de parcours,
- un passage à l'élément suivant.

Un objet d'une classe implantant l'interface **Iterator** permet de parcourir une structure de données. Pratiquement, un curseur sur une position (courante) précédant ou suivant un élément de la structure est conservé en mémoire (initialement avant le premier élément et à la fin après le dernier, entre deux éléments durant l'itération).

Méthode :

- **boolean hasNext()** : elle permet de tester l'existence d'un élément après le curseur ;
- **Object next()** : elle renvoie l'élément suivant et déplace le curseur après cet élément ;
- **void remove()** : elle supprime le dernier élément renvoyé par **next** (c'est-à-dire l'élément avant le curseur). Cette opération est optionnelle et peut se résumer à la levée de l'exception **UnsupportedOperationException.**
- **boolean hasPrevious()** qui teste s'il existe un élément avant le curseur ;
- **Object previous()** qui renvoie, s'il existe, l'élément avant le curseur ;
- **void add(Object o)** qui permet l'insertion d'un élément avant le curseur (c'est-à-dire entre les éléments qui seraient retournés par appel de **Object previous()** et **Object next()**) ;
- **int nextIndex()** qui renvoie l'index de l'élément avant le curseur ;
- **int previous** qui renvoie l'index de l'élément après le curseur ;
- **void set(Object o)** qui remplace le dernier élément retourné par **previous** ou **next** (si la dernière opération a modifié la liste, c'est-à-dire si la dernière opération a été **add** ou **remove**, l'exception **IllegalStateException** est levée.

Nous allons faire un schéma pour voir le fonctionnement de cette collection :

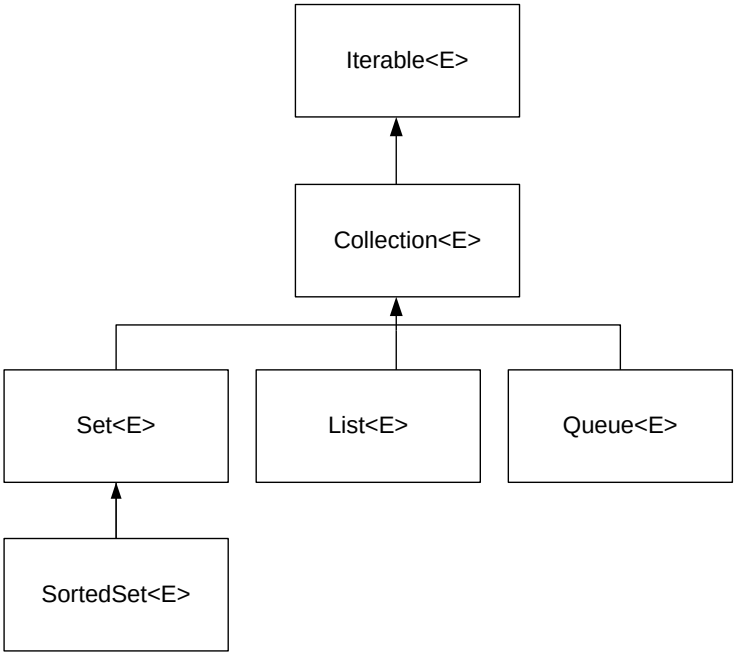


Figure 5: Diagramme des itérateurs

		Classes d'implantations			
		Table de hachage	Tableau	Arbre balancé	Liste chaînée
Interfaces	Set<E>	HashSet<E>		TreeSet<E>	
	List<E>		Array List<E>		LinkedList<E>
	Map<K,V>	HashMap<K,V>		TreeMap<K,V>	
	Queue<E>		Array Dequeue<E>		LinkedList<E>

Tableau 2 : rappel des différents itérateurs disponibles

Cette classe lance des exceptions non contrôlé :

- **next :** lance *NoSuchElementException* lorsqu'il n'y a plus rien à renvoyer.
- **remove :** lance *NoSuchOperationException* si la méthode n'est pas implémentée et lance IllegalException si *next* n'a pas été appelée avant ou si *remove* a déjà été appelée après le denier *next*.

On peut faire un petit rappel des fonctions utilisation iterator :

L'interface Collection<E> contient la méthode Iterator<E> Iterator qui renvoie un itérateur pour parcourir les éléments de la collection.

L'Iterator ne permet pas de la modifier mais que de parcourir.

Écriture avant JDK mais utile pour dcm4che2 :

```
for (Iterator it = coll.iterator(); it.hasNext(); ) {
        Employe e = (Employe)it.next();
        String nom = e.getNom(); }
```

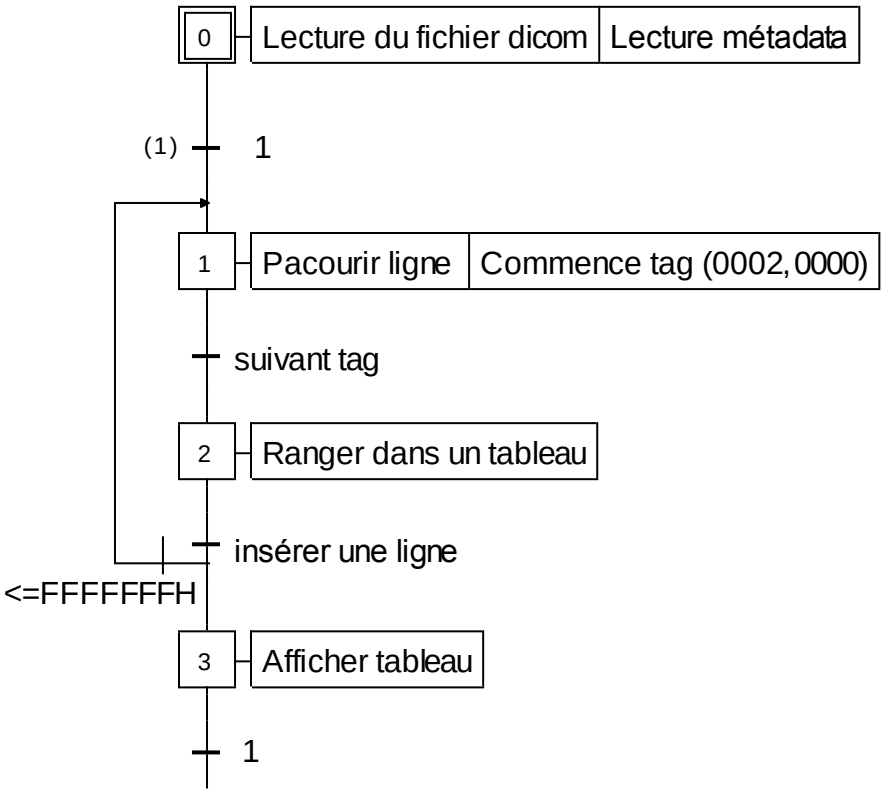
Ou

```
Collection collection = ...;
Iterator iterator = collection.iterator();
while (iterator.hasNext()) {
        Object element = iterator.next();
        if (condition(element)) {
        iterator.remove();
}
}
```

Dans le framework DCM4CHE 2:

- **Il faut parcourir les métadonnées, on veut les rangers dans un tableau, puis les imprimés.**

On a le schéma suivant qui explique le principe du codage.



Un itérateur qui garde les valeurs des données. La fonction nameOf de la méthode de la class DicomObject donne la description du tag en string. Le vrOf qui est une fonction qui retourne la valeur de la représentation courante de l'élément.

```
DicomInputStream dis = null;
try {
    dis = new DicomInputStream(selectedFile);
    DicomObject object = dis.readDicomObject();
    object.remove(Tag.PixelData);//supprime la lecture le tag PixelData, on ne rentre pas dedans
    Iterator<DicomElement> iter = object.iterator();//parcourt  de la liste des tags comme une file d'attente
    while (iter.hasNext()) {
        DicomElement element = iter.next();//prochain element dicom
        int tag = element.tag(); //déplacement dans le tag qui retourne un entier

        String tagName = ((DicomObject) object).nameOf(tag);//retourne me npm du tag
        String tagAddr = TagUtils.toString(tag);//retourne l'adresse du tag
        String tagVR = ((DicomObject) object).vrOf(tag).toString(); //retourne la valeur attribuer avec spécification
        /*Verification pour une image ultrason*/
        if (tagVR.equals("SQ")|element.hasDicomObjects()) {//tant que VR=SQ
            if (element.hasItems()) {//si il n'y a un item donc boucle
                affichageDicomItem(element.addDicomObject(element.getDicomObject()), i);
                i++;//inscrémentation de l'indice de la fenêtre
                continue;//on parcourt après l'item VR devient !SQ
            }
        }
        String tagValue = ((DicomObject) object).getString(tag);
        col.add(tagValue);
        System.out.println(tagAddr + "[" + tagVR + "]" + tagName + "[" + tagValue+ "]");
        model.insertRow(0, col);//rajouter chaque ligne
    }
} catch (IOException ex) {//on n'a pas charger l'element du serveur
    Logger.getLogger(Menu.class.getName()).log(Level.SEVERE, null, ex);
}
```

Méthode affichageDicomItem prend un DicomObject et I l'indice séquence.

```
public void affichageDicomItem(DicomObject object, int i) {


    try/*Condition d'erreur*/ {
        Iterator<DicomElement> iter = object.iterator();
       *Boucle qui permet permet de trouver les tags choisis*/
        while (iter.hasNext()) {
            Vector<String> col = new Vector<String>();//construction d'un vecteur tempon
            DicomElement element = iter.next();//prochain élèment dicom
            int tag = element.tag(); //déplacement dans le tag qui retourne un entier
            /*Vérification terminal*/
            String tagName = ((DicomObject) object).nameOf(tag);
            String tagAddr = TagUtils.toString(tag);//retourne l'adresse du tag
            String tagVR = ((DicomObject) object).vrOf(tag).toString(); //retourne la valeur attribuer avec
spécification


            String tagValue = ((DicomObject) object).getString(tag);
            System.out.println(tagAddr + "[" + tagVR + "]" + tagName + "[" + tagValue + "]");
                                    /*addictionnement des tags dans le tableau en colonne*/
        }
    } catch (Exception ex) {//erreur de chargement
         }//aucune erreur
}
```

III.2) Jouer avec les métadonnées :

a) Lire les métadonnées :

```
/**      * Read the file into a dicom object
         * @param f the dicom file
         * @return the read dicom object
         */
        public static DicomObject readDicomObject(File f) {
          DicomObject dcmObj;
          DicomInputStream din = null;
          try {
            din = new DicomInputStream(f);
            dcmObj = din.readDicomObject();
            return dcmObj;
          }
          catch (IOException e) {
                  e.printStackTrace();
            return null;
          }
          finally {
            try {
              din.close();
            }
            catch (IOException ignore) {
            }
          }
        }
```

Voir la source displayTag.java avec un grand nombre de méthode de gestion de métadonnées.

b) Lire les métadonnées :

Cette classe permet de vous montrer comment un tableau de tag peut afficher les informations de métadonnées :

```
package dicom;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import javax.swing.JFileChooser;
import org.dcm4che2.io.DicomInputStream;
import org.dcm4che2.data.DicomObject;
/**
 *
 * @author Dimitri
 */
public class test3 {
   public static final  int[] tag = {
                           0x00080020,
            0x00080022,
      };
private static String[] getValue(DicomObject object, int[]  PATIENT_ADDITIONAL_TAGS){
   String [] value = new String [PATIENT_ADDITIONAL_TAGS.length];
   int i =0;
   while (i<PATIENT_ADDITIONAL_TAGS.length){
                for (int tag : PATIENT_ADDITIONAL_TAGS) {

             value[i]=object.getString(tag);
            i++;
                }
        //System.out.print(value[0]+"\n");
//System.out.print(value[1]);
             }
    return value;
   }
public static void main(String[] args) throws FileNotFoundException, IOException{
       File fileInput = null, fileOutput= null;
             JFileChooser choix = new JFileChooser();
```

```
        int retour = choix.showOpenDialog(null);


        if(retour == JFileChooser.APPROVE_OPTION){
    fileInput = choix.getSelectedFile();
        JFileChooser saveDicom = new JFileChooser();
    saveDicom.setMultiSelectionEnabled(false);


    }


FileInputStream fis = new FileInputStream(fileInput);
DicomInputStream dis = new DicomInputStream(fis);
DicomObject obj = dis.readDicomObject();
String nounValue[] =getValue(obj,tag);
System.out.print(nounValue[0]+"\n");
System.out.print(nounValue[1]);
    }
}
```

c) Ecrire une métadonnée :

```
public static void writeDicomFile(DicomObject dObj, File f) {
            FileOutputStream fos;
            try {
               fos = new FileOutputStream(f);
            }
            catch (FileNotFoundException e) {
               e.printStackTrace();
               return;
            }
            BufferedOutputStream bos = new BufferedOutputStream(fos);
            DicomOutputStream dos = new DicomOutputStream(bos);
            try {
               dos.writeDicomFile(dObj);
            }
            catch (IOException e) {
               e.printStackTrace();
               return;
```

```
        }
        finally {
            try {
                dos.close();
            }
            catch (IOException ignore) {
            }
        }
    }
```

Voir la class displayTag.java.

III.3) Gestion des images DICOM :

Nous allons voir comment nous pouvons ouvrir une image dicom. Nous allons dans un premier temps se rappeler les différentes caractéristiques d'une image dicom. Puis pour une image classique en java puis avec le pack dcm4che en expliquant les différentes fonctions issues du code source.

a) Explication DICOM :

On ce rappelle que la structure du fichier :

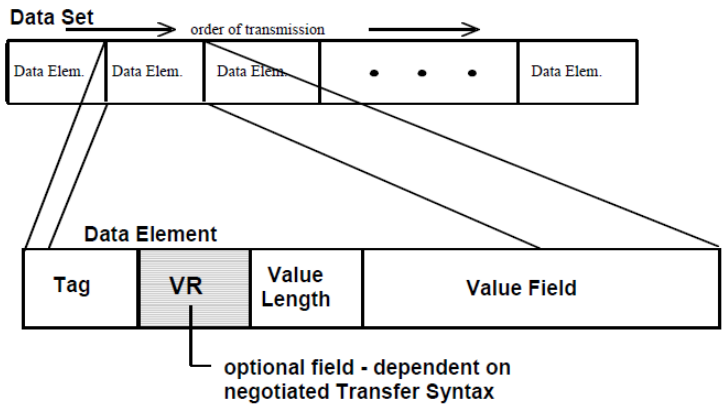


Figure 6 : schéma de l'étiquette DICOM Data Element

Nous avons pour image DICOM les tags suivants utiles :

TAG(groups, element)

- 0028 diplay data
- 7FE0 image data : le tag où nous trouvons l'ensemble de la matrice pixel

Codage :

DICOM : entre 8 à 16 bits par pixel →sort 2 octets par pixel

Standart : 8 bits → sort 1 octet par pixel

Ce standard sort trois donnés spécifiques :

- Bits allocated (0028, 0100) : le nombre de bit par autre pixel, qui doit être seulemenr de 9 ou 16.
- Bits stored (0028, 0010) : le nombre de bits qui renferme les informations de l'image.
- High bit (0028, 0102) : donne la position de plus significatif des bits (MSB)
- Les Samples par Pixel DE (0028, 0002) définies le nombre de couleur (color channel) par pixel. Gray-scale image a une valeur de 1, RGB imaegs de 3.
- La photometric Interpretation DE (0028, 0004) stores information about how the pixel data is to be interpreted.
    - MONOCHROME 1 : gray-scale image avec noir comme le maximum de valeur et blanc comme le minimum.
    - MONOCHROM 2 : gray-scale image avec blanc comme le maximum valeur et noir est comme minimum.
    - PALETTE COLOR : image couleur utilise LUT (sample per pixel = 1).
        - The respective tags are :
            - (0028, 1201) : red
            - (0028, 1202) : green
            - (0028, 1203) : blue
        - RGB : color image ; sample per pixel = 3
        - HSV: color image; sample per pixel = 3
        - ARGB : RGB image avec un additionnement d'alpha channel containing plus l'information ou overlays, sample per pixel = 4.
        - CMYK : color image ; samples per pixel = 4
        - YBR_FULL: color image using YC_bC_r channels; samples per pixel = 3
        - YBR_FULL_422: same as above with 4:2:2 color sampling

b) Pour une image JPEG en java :

b.1) *BUFFEREDIMAGE*

Il faut utiliser un BUFFEREDIMAGE pour mettre en tampon une image pixel.

La classe **java.awt.image.BufferedImage** hérite de **java.awt.Image** et correspond à un tableau rectangulaire de pixels.

– A chaque pixel est associé la couleur d'un point, dans une parmi plusieurs formes possibles appelées ***valeurs d'échantillonage***.

- Ces valeurs sont interprétées selon le ***modèle de couleur*** de l'image (**ColorModel**).

Un **BufferedImage** est composé :

– d'un **ColorModel** qui définit la façon d'interpréter les couleurs

– d'un **WritableRaster**, donc d'un raster autorisé en écriture.

Un **Raster** est composé :

– d'un **DataBuffer** contenant les données brutes, dans un tableau

– d'un **SampleModel** (modèle d'échantillonnage) qui interprète les données brutes.

Création d'une image vide

BufferedImage(int width, int height, int typeImage)

Les types d'images indiquent comment les couleurs des pixels sont codées.

TYPE_3BYTE_BGR	bleu, vert, rouge, sur 8 bits chacun
TYPE_4BYTE_ABGR	alpha, bleu, vert, rouge, sur 8 bits chacun
TYPE_4BYTE_ABGR_PRE	alpha, bleu, vert, rouge, sur 8 bits chacun, les couleurs pondérées
TYPE_BYTE_BINARY	1 bit par pixel, groupés en octets
TYPE_BYTE_INDEXED	1 octet par pixel, indice dans une table de couleurs
TYPE_BYTE_GRAY	1 octet par pixel, niveau de gris
TYPE_USHORT_555_RGB	rouge, vert, bleu, sur 5 bits, codés dans un short
TYPE_USHORT_565_RGB	rouge sur 5, vert sur 6, bleu sur 5 bits
TYPE_USHORT_GRAY	niveau de gris sur 16 bits
TYPE_INT_RGB	rouge, vert, bleu sur 8 bits chacun, dans un int
TYPE_INT_BGR	bleu, vert, rouge (Solaris)
TYPE_INT_ARGB	alpha, rouge, vert, bleu sur 8 bits, dans un int
TYPE_INT_ARGB_PRE	les couleurs déjà pondérées par alpha

b.2) WritableRaster :

- Permet d'accéder à un pixel :

```
BufferedImage image=new BufferedImage(500,400,
BufferedImage.TYPE_AINT_RGB);
WritableRaster raster = image.getRaster();
```

- Pour changer la valeur des pixels - la méthode générique **raster.setDataElement(int x,int y,Object datas)**, où datas est la valeur renvoyée par **getDataElement()**
- une des méthode **raster.setPixel()**

```
Object datas = model.getDataElements(Color.BLUE.getRGB(), null);
raster.setDataElement(i,j,datas);
```

b.3) setPixel() :

Cette méthode permet de surchagée chaque type d'image.

● Pour le type **TYPE_INT_ARGB**, la méthode est **raster.setPixel(int i,int j,int[] datas)** le tableau doit contenir 4 entiers entre 0 et 255, pour les composantes alpha, rouge, vert, bleu :

```
int[] blue = {0,0,0,255};
raster.setPixel(i,j,blue);
```

b.4) Changement de type:

Pour changer une image du type à un autre, le plus simple consiste à copier l'image dans une autre créé avec le bon type :

```
private static BufferedImage copyImage(BufferedImage src, int imageType) {
        BufferedImage bufferedImage=new BufferedImage(
        src.getWidth(),src.getHeight(), imageType);
        Graphics2D g2 = bufferedImage.createGraphics();
        g2.drawImage(src, null, null);
        g2.dispose();
return bufferedImage;
}
```

Pour copier une image dans une autre, le plus facile est de passer par un **Graphics.**

b.5) Changement d'une image:

- En Java, les images peuvent être chargées à partir : d'un nom de fichier, d'une URL, d'un flux (InputStream)

Il existe deux façons de charger des images :

- La classe java.awt.Toolkit (depuis la 1.0), permet uniquement de lire les images, utilise les décodeurs présent sur la plateforme, au moins GIF, JPG et PNG (depuis la 1.3)
- La classe javax.imageio.ImageIO (depuis la 1.4), permet de lire et écrire, possède un mécanisme de SPI
-

● ImageIO possède 4 méthodes read qui renvoie un BufferedReader en fonction de d'une URL, d'un File, d'un InputStream ou d'un ImageInputStream

● ImageIO possède 3 méthodes read qui prennent en paramétre un BufferedReader, un nom de format, et soit un File, un OutputStream ou un ImageOutputStream :

```
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
public class Transcode {
public static void main(String[] args) throws IOException {
        BufferedImage image=ImageIO.read(new File("lena.jpg"));
        ImageIO.write(image,"png",new File("lena.png"));
}
}
```

b.6) décoder une image :

La méthode **ImageIO.getReaderFormatNames()** permet de connaître l'ensemble des formats supportés et **getImageReadersByFormatName(String formatName)** renvoie un iterateur sur l'ensemble des ImageReader pour un format :

```
public class AllReaders {
   public static void main(String[] args) throws IOException {
        for(final String format:ImageIO.getReaderFormatNames()) {
             for(ImageReader reader:new Iterable<ImageReader>() {
                public Iterator<ImageReader> iterator() {
               return ImageIO.getImageReadersByFormatName(format);
                                                          }
                                                 })
System.out.println(format+" "+reader.getClass().getName());
                                          }
          }
}
```

b.7) créer une image :

- **getImage()** par raport à **createImage()** maintient un cache de toutes les images chargées pour partager la même image entre plusieurs appels
- Le problème est que ce cache ne libère que rarement (voir jamais) les images chargées, donc **à ne pas utiliser**
-

```
public static Image getImage(String filename){
        Image image=cache.get(filename);
if (image==null) {
    image=createImage(filename);
        cache.put(filename,image);
}
return image;
}
    private final Map<String,Image> cache=...
```

c) Pour une image DICOM en java :

c.1) Rappel des différents package :

org.dcm4che2.image :

Class Summary	
ByteLookupTable	
ColorModelFactory	
LookupTable	
OverlayUtils	Provides utility methods to extract overlay information from DICOM files.
ShortLookupTable	
SimpleYBRColorSpace	
VOIUtils	

org.dcm4che2.imageio :

Class Summary	
ImageReaderFactory	
ImageWriterFactory	
ItemParser	
ItemParser.Item	

c.2) La création d'une image :

```
File sourceFile = new File("cardio.dcm");
```

```
                ImageInputStream in =
ImageIO.createImageInputStream(sourceFile);//On crée une entreé de bit
```

c.3) Chargement d'une image :

1er étape : on crée une liste donc un tableau de valeur de bit de pixel, pour permettre de ce déplacé dans les pixels.

On sait qu'une image dicom est codée selon cette façon (voir norme dicom) :

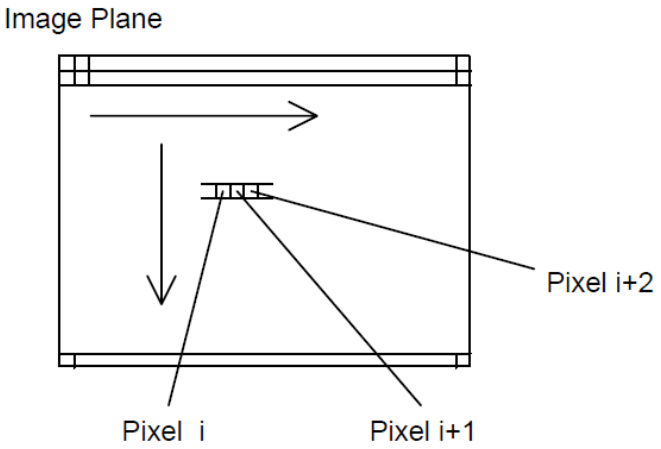


Figure 6: Déplacement des pixels

Puis

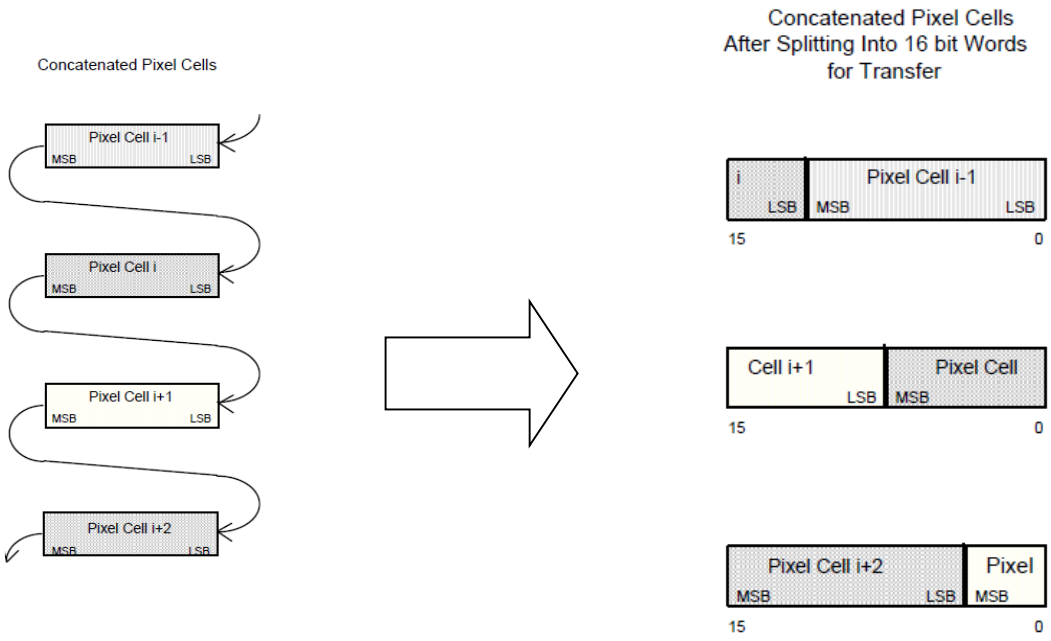


Figure 7: Principe en détail de l'image dicom

On peut voir qu'il faut ce déplacé dans les pixels, donc il faut créer un itérateur pour permettre de parcourir l'ensemble des pixels de l'image.

Il sera donc normal de faire une itération :

Iterator<ImageReader> iter = imageIO.*getImageReadersByFormatName*("DICOM");

// returns an Iterator containing all currently registered ImageReaders that claim to be able to decode the named format.

```
ImageReader reader = (ImageReader) iter.next();
```

On veut ensuite récupéré les différents paramètres dicom alors les lires :

Class DicomImageReader méthode getDefaultReadParam qui retourne DicomImageReadParam();

DicomImageReadParam param = (DicomImageReadParam) reader.getDefaultReadParam();

La méthode getDefaultParam renvoie les paramètres de l'images sources : la hauteur, longueur, et add a Pixel Data attribute with the byte array from the DataBuffer of the scaled Raster.

- *1er point : class DicomImageReadParam*

Nous allons rentrer dans la fonction pakage open-source :

Java.lang.string	getOverLayRGB()	Get the 6 digit hex string that specifies the RGB colour to use for the overlay
DicomObject	getPresentationState()	@return DicomObject prState
Short[]	getPValue2Gray()	@return short[] pval2gray
DicomObject	getVoiLut()	@return DicomObject voiLut
Java.lang.String	getVoiLutFonction()	@return String vlutFct
float	getWindowCenter()	@return center
float	getWindowWidth()	@return width
boolean	isAutoWindowing()	Si c'est une image qui retourne une taille de l'image automatiquement @return true sinon @return false
void	setAutoWindowing(boolean autoWindowing)	@return autoWindowing
void	setOverLayRGB(java.lang.String overlayRGB)	Sets the 6 digit hex string that specifies the RGB colours to use for overlay.

void	setPresentationState(DicomObject prostate)	Construit
void	setValue2Gray(short[] pval2gray)	Construit un tableau de short de pval2gray.
void	setVoilLut(DicomObject voiLut)	Créer le vluFct en entrée un DicomObject.
void	setVoiLutFonction(java.lang.String vluFct)	Créer le vluFct en entrée un string.
void	setWindowCenter(float center)	Créer le centre de l'image.
void	setWindowWidth(float width)	Créer le width de l'image.

Fin du 1er point

On crée ensuite un tampon de l'image.

```
ImageInputStream iis = ImageIO.createImageInputStream(file);
```

On récupère le décalage les pixels :

```
reader.setInput(in, true); //permet de ce décaler dans les pixels
                  //Sets the input source to use to the given ImageInputStream
or orther Object. The input source must be set before any of the query or
read methods are used If input is null, any currently
                  //set input source will be removed. In any case, the
value of minIndex will be initialized to 0.
               //si true, image et les metadata peut être lu pour une image
source
```

```
Il faut maintenant récupérer le nombre de frame dans l'image dicom :
```

```
int number = readers.getNumImages(true);//numberOfFrame on a "readers" qui
doit être DicomImage
```

En fin nous voulons mettre dans un BufferedImage pour permettre de la lire et de récupérer les différents paramètres.

Syntaxe:

```
public java.awt.image.BufferedImage read(int imageIndex,
javax.imageio.ImageReadParam param)
```

Signification:

Reads the provided image as a buffered image. It is possible to read image overlays by providing the 0x60000000 number associated with the overlay. Otherwise, the imageIndex must be in the range 0...numberOfFrames-1, or 0 for a single frame image. Overlays can be read from PR objects or other types of objects in addition to image objects. param can be used to sepecify GSPS to apply to the image, or to override the default window level values, or to return the raw image.

- *2ème point : read permet de faire la décompression*

```
/**
 * Reads the provided image as a buffered image. It is possible to read
 * image overlays by providing the 0x60000000 number associated with the
 * overlay. Otherwise, the imageIndex must be in the range
 * 0..numberOfFrames-1, or 0 for a single frame image. Overlays can be read
 * from PR objects or other types of objects in addition to image objects.
 * param can be used to sepecify GSPS to apply to the image, or to override
 * the default window level values, or to return the raw image.
 */
@Override
public BufferedImage read(int imageIndex, ImageReadParam param)
        throws IOException {
    if (OverlayUtils.isOverlay(imageIndex)) {
        readMetaData();
        String rgbs = (param != null) ? ((DicomImageReadParam) param)
                .getOverlayRGB() : null;
        return OverlayUtils.extractOverlay(ds, imageIndex, this, rgbs);
    }

    initImageReader(imageIndex);
    if (param == null) {
        param = getDefaultReadParam();
    }

    BufferedImage bi;
```

Permet de trouver les paramètres rgb.

Si les paramètres de l'image (hauteur, taille, nbre de pisxel) est égale à 0 alors on renvoie la méthode getDefaultReadParam().

```
if (compressed) {
    ImageReadParam param1 = reader.getDefaultReadParam();
    copyReadParam(param, param1);
    bi = reader.read(0, param1);
    postDecompress();
} else if( pmi.endsWith("422") || pmi.endsWith("420") ) {
    WritableRaster wr = (WritableRaster) readRaster(imageIndex, param);
    bi = new BufferedImage(ColorModelFactory.createColorModel(ds),
            wr, false, null);
} else {
    bi = reader.read(imageIndex, param);
    if (swapByteOrder) {
        ByteUtils.toggleShortEndian(
                ((DataBufferByte)bi.getRaster().getDataBuffer()).getData());
    }
}
if (monochrome) {
    WritableRaster raster = bi.getRaster();
    LookupTable lut = createLut((DicomImageReadParam) param,
            imageIndex + 1, raster);
    if (lut != null) {
        WritableRaster dest = raster;
        if (dest.getDataBuffer().getDataType() != DataBuffer.TYPE_BYTE
                && (lut instanceof ByteLookupTable)) {
            BufferedImage ret = new BufferedImage(bi.getWidth(), bi
                    .getHeight(), BufferedImage.TYPE_BYTE_GRAY);
            dest = ret.getRaster();
            bi = ret;
        }
        DataBuffer destData = dest.getDataBuffer();
        lut.lookup(raster, dest);
        if (destData.getDataType() == DataBuffer.TYPE_SHORT) {
            ColorModel cm = bi.getColorModel();
            short[] ss = ((DataBufferShort) destData).getData();
            return new BufferedImage(cm, Raster.createWritableRaster(
                    raster.getSampleModel(), new DataBufferUShort(ss,
                            ss.length), null),
                    cm.isAlphaPremultiplied(),
                    new Hashtable<Object,Object>());
```

Permet de voir la compression du format dicom. Puis de donner les caractères par défaut des images.

Conversion de la couleur avec le monochrome.

```
            }
          }
        }
        return bi;
    }       return bi;
    }
```

fin du point 2

Remarque de ne pas oublier d'installer le package jai-io sur votre ordinateur et mettre les deux points jar dans le fichier lib de dcm4che2.

Car le dicom utilise plusieurs format d'image et la fonction read(int imageIndex, javax.iamgeio.ImageReadParam param).

Je justifie dans le fichier du package de la class DicomImageReader, il import import **com.sun.media.imageio.stream.RawImageInputStream ;**

&import com.sun.media.imageio.stream.SegmentedImageInputStream;

Et ils ont implementé une variable qui s'appelle J2KIMAGE_READER = "com.sun.media.imageioimpl.plugins.jpeg2000.J2KImageReader";

C *3ème point : fichier qui permet de vérifier le dicom avec son extension*

```
package org.dcm4che2.imageioimpl.plugins.dcm;

import java.io.EOFException;
import java.io.IOException;
import java.util.Locale;

import javax.imageio.ImageReader;
import javax.imageio.spi.ImageReaderSpi;
import javax.imageio.stream.ImageInputStream;

public class DicomImageReaderSpi extends ImageReaderSpi {

/*instruction des extensions dicom*/
```

```
private static final String[] formatNames = { "dicom", "DICOM" };
private static final String[] suffixes = { "dcm", "dic", "dicm", "dicom" };
private static final String[] MIMETypes = { "application/dicom" };
private static String vendor;
private static String version;
static {
    Package p = DicomImageReaderSpi.class.getPackage();
    vendor = maskNull(p.getImplementationVendor(), "");
    version = maskNull(p.getImplementationVersion(), "");
}

private static String maskNull(String s, String def) {
    return s != null ? s : def;
}

public DicomImageReaderSpi() {
    this(vendor, version, MIMETypes,
            "org.dcm4che2.imageioimpl.plugins.dcm.DicomImageReader",
            STANDARD_INPUT_TYPE, null, false, false);
}

/**
 * A constructor added to easier extend this class.
 */
protected DicomImageReaderSpi(String vendorName, String version,
        String[] MIMETypes, String readerClassName, Class[] inputTypes,
        String[] writerSpiNames,
        boolean supportsStandardStreamMetadataFormat,
        boolean supportsStandardImageMetadataFormat) {

    super(vendorName, version, formatNames, suffixes, MIMETypes,
            readerClassName, inputTypes, writerSpiNames,
            supportsStandardStreamMetadataFormat, null, null, null, null,
            supportsStandardImageMetadataFormat, null, null, null, null);
```

```
    }

    public String getDescription(Locale locale) {
        return "DICOM Image Reader";
    }

    public boolean canDecodeInput(Object input) throws IOException {
        if (!(input instanceof ImageInputStream)) {
            return false;
        }

/*Convertir de l'image à l'entrée*/
        ImageInputStream stream = (ImageInputStream) input;
        byte[] b = new byte[132];
        stream.mark();
        try {
            stream.readFully(b);
        } catch (EOFException e) {
            return false;
        } finally {
            stream.reset();
        }
        if (b[128] == 0x44 // D
                && b[129] == 0x49 // I
                && b[130] == 0x43 // C
                && b[131] == 0x4D) { // M
            return true;
        }
        try {
            if (b[0] == 0) { // big endian
                if (b[1] == 0) {
                    return false;
                }
                int len = ((b[6] & 0xff) << 8) | (b[7] & 0xff);
```

```
                return (b[1] == b[len + 9]);
            }

            // little endian
            if (b[1] != 0) { // expect group tag <= 00FF
                return false;
            }
            int len = (b[6] & 0xff) | ((b[7] & 0xff) << 8);
            if (b[0] == b[len + 8]) {
                return true;
            }
            len = (b[4] & 0xff) | ((b[5] & 0xff) << 8);
            return (b[0] == b[len + 8]);
        } catch (IndexOutOfBoundsException e) {
            return false;
        }
    }


    public ImageReader createReaderInstance(Object extension) {
        return new DicomImageReader(this);
    }
}
```

- *4ème point : explication de numOfFrame*

On se rappelle que number of Frame est sur le tag (0028,0010).

Permet de trouver le nombre de fichier présent dans l'image.

Il existe dans le fichier du package DicomImageReader des fonctions suivantes pour cette analyse :

```
public int getNumImages(boolean allowSearch) throws IOException {
    readMetaData();
    return frames;
```

```
}
```

Cette fonction permet de rappeler une fonction readMetaData() qui renvoie un entier :

i. vérifier s'il y a un fichier possible à ouvrir
ii. lecture du méta-data

On peut voir qu'on doit :

1. lire le fichier entrée : DicomInputStream dis = new DicomInputStream(nomFichier)
2. ouverture des informations dans le fichier par la classe DicomObject
3. supprimer le tag PixelData : on applique l'instruction suivante :

dis.setHandler(new StopTagInputHandler(Tag.PixelData)) ;

→L'instruction StopTagInputHandler permet d'arrêter l'itération des tags avant celui choisi :

- Lecture des métadata : class DicomStreamData
  - Constructeur DicomStreamData
  - On veut voir les métadata du fichier :

Méthode de cette classe : public final void setDicomObject(DicomObject dataset). → la variable dataset prend le fichier lu par DicomObject ;

- Transformation des données en octets :

```
bigEndian = dis.getTransferSyntax().bigEndian();
```

- Tsuid, width, height, frame, allocated,

ds.getString(int tag) : @return le nom du tag

ds.getInt(int tag);@renvoie la valeur entière d'un tag

- Si le tag est PixelData pour un frame (donc numberOfFrame) donc on lui renvoie 1. Puis on affiche les différents paramètres.

```
if (dis.tag() == Tag.PixelData) {
        if (frames == 0)
           frames = 1;
        swapByteOrder = bigEndian && dis.vr() == VR.OW
             && dataType == DataBuffer.TYPE_BYTE;
        if (swapByteOrder && banded) {
           throw new UnsupportedOperationException(
                "Big Endian color-by-plane with Pixel Data VR=OW not implemented");
        }
```

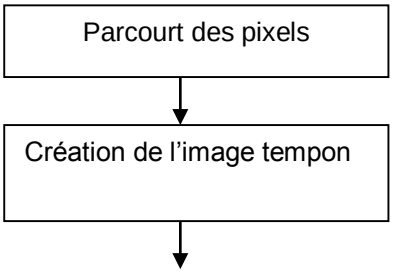
Voir ci-dessous la fonction complète :

```
private void readMetaData() throws IOException {
        if (iis == null) {
            throw new IllegalStateException("Input not set!");
        }
        if (ds != null) {
            return;
        }
        dis = new DicomInputStream(iis);
        dis.setHandler(new StopTagInputHandler(Tag.PixelData));
        ds = dis.readDicomObject();
        streamMetaData = new DicomStreamMetaData();
        streamMetaData.setDicomObject(ds);
        bigEndian = dis.getTransferSyntax().bigEndian();
        tsuid = ds.getString(Tag.TransferSyntaxUID);
        width = ds.getInt(Tag.Columns);
        height = ds.getInt(Tag.Rows);
        frames = ds.getInt(Tag.NumberOfFrames);
        allocated = ds.getInt(Tag.BitsAllocated, 8);
        banded = ds.getInt(Tag.PlanarConfiguration) != 0;
        dataType = allocated <= 8 ? DataBuffer.TYPE_BYTE
                : DataBuffer.TYPE_USHORT;
        samples = ds.getInt(Tag.SamplesPerPixel, 1);
        monochrome = ColorModelFactory.isMonochrome(ds);
        pmi = ds.getString(Tag.PhotometricInterpretation);

        if (dis.tag() == Tag.PixelData) {
            if (frames == 0)
                frames = 1;
            swapByteOrder = bigEndian && dis.vr() == VR.OW
                    && dataType == DataBuffer.TYPE_BYTE;
            if (swapByteOrder && banded) {
                throw new UnsupportedOperationException(
                        "Big Endian color-by-plane with Pixel Data VR=OW not implemented");
```

```
        }
        pixelDataPos = dis.getStreamPosition();
        pixelDataLen = dis.valueLength();
        compressed = pixelDataLen == -1;
        if (compressed) {
           ImageReaderFactory f = ImageReaderFactory.getInstance();
           log.debug("Transfer syntax for image is " + tsuid
                 + " with image reader class " + f.getClass());
           f.adjustDatasetForTransferSyntax(ds, tsuid);
        }
     }
  }
```

c.4) Le programme de l'affichage :

Un petit diagramme d'explication :



Lecture des données pour les décompressés (parcourt des images dans frame et paramètre de l'image-hauteur, largeur, nbre pixel-)

```
public BufferedImage chargeImageDicomBufferise(File file, int value) throws IOException  {


                        Iterator<ImageReader> iter =
ImageIO.getImageReadersByFormatName("DICOM");//spécifie l'image



                        ImageReader readers = (ImageReader)iter.next();//on se déplace dans
l'image dicom


                    DicomImageReadParam  param1 =  (DicomImageReadParam)
readers.getDefaultReadParam();//return DicomImageReadParam

                     //    Adjust the values of Rows and Columns in it and add a Pixel Data attribute
with the byte array from the DataBuffer of the scaled Raster


                        ImageInputStream iis = ImageIO.createImageInputStream(file);


                    readers.setInput(iis, true);//sets the input source to use the given
ImageInputSteam or other  Object


                        BufferedImage image = readers.read(value, param1);//BufferedImage image
= reader.read(frameNumber, param); frameNumber = int qui est l'imageIndex

                        System.out.println(image);//affichage au terminal des caractères de l'image


                        readers.dispose();//Releases all of the native sreen resources used by this
Window, its subcomponents, and all of its owned children

                    return this.image = image;}
```

Pour affichage de l'overlay, on fait alors : (source Dicom.java)

```
public static BufferedImage overlayDicom (File file) throws FileNotFoundException, IOException{

   FileInputStream fis = new FileInputStream(file);

   DicomInputStream dis = new DicomInputStream(fis);

   DicomObject obj = dis.readDicomObject();
```

```
   Iterator<ImageReader> iter = ImageIO.getImageReadersByFormatName("DICOM");//spécifie
l'image

   ImageReader readers = iter.next();

   if(obj.contains(Tag.OverlayData)==true ){

   BufferedImage buf = OverlayUtils.extractOverlay(obj,Tag.OverlayData , readers,null);

   return buf;


} else {

    return null;

     }




}
```

III.4) Le raster:

Pseudo code :

```
      //ouvrir ce fichier avec vérification de l'extension dicom

      Iterator<ImageReader> iter = ImageIO.getImageReadersByFormatName("DICOM");


       //obtenir ses données de pixel

       DicomImageReader readers = (DicomImageReader)iter.next();();//on se deplace dans l'image
dicom (dans les pixels)


      // Adjust the values of Rows and Columns in it and add a Pixel Data
attribute with the byte array from the DataBuffer of the scaled Raster


    DicomImageReadParam  param =  (DicomImageReadParam) readers.getDefaultReadParam();


      //Raster représente les valeurs des pixels rectangulaires occupant un
domaine particulier (lire tableau de pixels)

ImageInputStream iis = ImageIO.createImageInputStream(file); // créer un fichier entrée


      //sets the input source to use the given ImageInputSteam or other
Object
```

readers.setInput(iis, **true**);//

Le code complet:

```
/**
 * Permet d'avoir le raster d'une image
 * @param file : fichier courant
 * @param indice : l'indice du frame
 * @return un raster de l'image
 * @throws IOException
 */
public Raster rasterPicture(File file, int indice) throws IOException{

        //ouvrir ce fichier et obtenir ses données de pixel
         Iterator<ImageReader> iter = ImageIO.getImageReadersByFormatName("DICOM");//specifie l'image

         DicomImageReader readers = (DicomImageReader)iter.next();//on se deplace dans l'image dicom

        DicomImageReadParam  param =  (DicomImageReadParam) readers.getDefaultReadParam();//return DicomImageReadParam

         //    Adjust the values of Rows and Columns in it and add a Pixel Data attribute with the byte array from the DataBuffer of the scaled Raster

         ImageInputStream iis = ImageIO.createImageInputStream(file);

        readers.setInput(iis, true);//sets the input source to use the given ImageInputSteam or other  Object
        //Lu les pixels

//Raster représente les valeurs des pixels rectangulaires occupant un domaine particulier (lire tableau de pixels)
         // Alors, si notre objet raster est nul nous avons obtenu quelques erreurs l'ouverture du fichier

        Raster myImage = readers.readRaster(indice, param);  //lecture des données de l'images
```

```
            System.out.print(myImage);// renvoie les caracteristique que de l'image


           /*Recupère des donnees */
            setWidth(readers.getWidth(indice));
            setHeight(readers.getHeight(indice));


          return this.setMyImage(myImage);
        }
```

III.5) Lecture des tags :

Pseudo code :

→ Lecture du fichier dicom : DicomInputStream dis = **new** DicomInputStream(file);

→ Lire les métadatas d'un dicom : DicomObject object = dis.readDicomObject();

→ Parcourir l'ensemble des tags : Iterator<DicomElement> iter = object.iterator();//deplacement dans les tags

→ Boucle :

Tant que itérator n'a pas finit

- la prochaine ligne de métadatas : DicomElement element= iter.next();
- ce déplacé dans les tags (group, element) : **int** tag = element.tag();
- retourner le VR (Value reference) : String tagVR = ((DicomObject) object).vrOf(tag).toString();

Si VR = SQ ou si nous avons pu parcourir toute les métadonnées dans le fichier dicom (cas des tags spécifiques du constructeur)

Si nous avons parcourus toute les idems alors nous devons refaire une lecture dans les métadonnées.

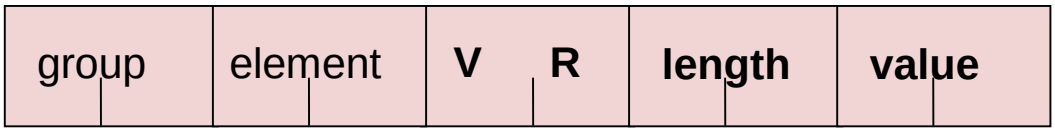
```
if (tagVR.equals("SQ")| element.hasDicomObjects()){
                                    if(element.hasItems() ){//si il n'y a un item donc boucle
                                            element.addDicomObject(element.getDicomObject());

                                            continue;//passe cette condition pour lire la fin des tags
                                    }
                            }
```

Nous allons expliquer se métadonnée specifique:

Il existe dans le standart dicom dans la partie 5 l'explication de phénomène.

Nous reprenons notre schéma de départ pour les métadonnées.

Lorque que le VR est égale à SQ, on a le phénomène suivant :



Lorque nous arrivons à un tag qui a le VR qui a SQ, nous avons la structure suivante :

Le group et element ne change pas mais nous aurons la value une nouvelle séquence métadata spécifique par exemple pour les ultrasons.

On refait dans value soit plusieurs items qui est structuré comme suivant :

Le tag de l'item, la longueur, la valeur

Data Element Tag	Value Representation		Data Element Length	Data Element Value							
(gggg, eeee) with VR of SQ	SQ	0000H Reserved	FFFF FFFFH un-defined length	First Item			Second Item			Sequence Delimitation Item	
				Item Tag (FFFE, E000)	Item Length 98A5 2C68H	Item Value Data Set	Item Tag (FFFE, E000)	Item Length B321 762CH	Item Value Data Set	Seq. Delim. Tag (FFFE, E0DD)	Item Length 0000 0000H
4 bytes	2 bytes	2 bytes	4 bytes	4 bytes	4 bytes	98A5 2C68H bytes	4 bytes	4 bytes	B321 762CH bytes	4 bytes	4 bytes

Le code complet :

```
/**
        * RechercheDicomItem permet de ce déplacé dans le tag pour accéder à item
        * Vérifie si le tag existe dans les métas datas
        * @param file : fichier entrée
        * @param TAG : valeur du tag selon dcm4che2
        * @throws IOException
        */
            public static DicomObject rechercheTagDicom (File file) throws IOException {
                /*vérification des extensions dicom*/

                DicomInputStream dis = new DicomInputStream (file);
                DicomObject object = dis.readDicomObject();//lit les metadatas du fichier
dicom
```

```
// object.remove(Tag.PixelData);//on enlève le pixelData

Iterator<DicomElement> iter = object.iterator();//deplacement dans les tags

while (iter.hasNext ()) {

DicomElement element=  iter.next ();//prochain element dicom
int tag = element.tag (); //déplacement dans le tag qui retourne un entier

String tagVR =  ((DicomObject) object).vrOf(tag).toString();
//retourne la valeur attribuer avec spécification

/*Lecture des metadatas dans items*/
if (tagVR.equals("SQ")| element.hasDicomObjects()){
if(element.hasItems() ){//si il n'y a un item donc boucle
element.addDicomObject(element.getDicomObject());

continue;//passe cette condition pour lire la fin des tags
}
}

}
return object;

}
```

III.6) Réalisation d'un filtre :

- fabrication de la matrice pour le filtre :

```
float[] emboss = new float[] { -2,0,0,   0,1,0,   0,0,2 };
```

- The Kernel class defines a matrix that describes how a specified pixel and its surrounding pixels affect the value computed for the pixel's position in the output image of a filtering operation. The X origin and Y origin indicate the kernel matrix element that corresponds to the pixel position for which an output value is being computed.

```
Kernel kernel = new Kernel(3, 3, emboss);
```

- Constructs a ConvolveOp given a Kernel.

```
ConvolveOp op = new ConvolveOp(kernel);
```

- Création d'un nouveau raster pour le filtre

```
Raster newRaster = op.filter(getMyImage(), null);
```

- Lecture des pixels pour une échelle de gris de 16 bits – 2 octets comme pour le tag de dicom

```
DataBufferUShort buffer = (DataBufferUShort)
newRaster.getDataBuffer();
```

- Retouner le tableau pour le premier bloc

```
short[] pixelData = buffer.getData();
```

- Met les nouveaux array dans le métadata PixelData

```
dcm.putShorts(Tag.PixelData, dcm.vrOf(Tag.PixelData), pixelData);
```

- Réation d'un nouveau fichier dicom pour un seul frame

```
FileOutputStream fos = new FileOutputStream(new File(fileOutput+".dcm"));
BufferedOutputStream bos = new BufferedOutputStream(fos);
DicomOutputStream dos = new DicomOutputStream(bos);
dos.writeDicomFile(dcm);  //creation du fichier dicom
dos.close();
```

Le code complet:

```
public DicomImageProcessing2(final File fileInput, final int indice, String
fileOutput) {
            try {
                  //DicomInputStream dis = new DicomInputStream(fileInput);
                  dcm = rechercheTagDicom(fileInput);
            } catch (IOException e1) {
                  // TODO Auto-generated catch block
                  e1.printStackTrace();
            }//lit les metadatas du fichier dicom
```

```
            try {
                  rasterPicture(fileInput, indice);
                  } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                  }
               float[] emboss = new float[] { -2,0,0,    0,1,0,    0,0,2 };


                  Kernel kernel = new Kernel(3, 3, emboss);
                 ConvolveOp op = new ConvolveOp(kernel);
                 Raster newRaster = op.filter(getMyImage(), null);
                //To extract the array
                 DataBufferUShort buffer = (DataBufferUShort)
newRaster.getDataBuffer();
                 short[] pixelData = buffer.getData();


                 //create a copy of our Dicom file without pixel data
                   try {
                        //dsmd = (DicomStreamMetaData)
reader.getStreamMetadata();
                        // DicomObject dicom = dsmd.getDicomObject();


                         dcm.putShorts(Tag.PixelData,
dcm.vrOf(Tag.PixelData), pixelData);  //changement du tag de dicom, sur le
Pixel data
                         FileOutputStream fos = new FileOutputStream(new
File(fileOutput+".dcm"));
                         BufferedOutputStream bos = new
BufferedOutputStream(fos);
                         DicomOutputStream dos = new
DicomOutputStream(bos);
                         dos.writeDicomFile(dcm);
                         dos.close();
                   } catch (IOException e) {
                        System.out.println("error de fichier");
                        e.printStackTrace();
                   }  }
```

Table ASCII

OCTAL	DÉCIMAL	HEXADÉCIMAL	CARACTÈRE	NOM
000	0	00	NUL	
001	1	01	SOH	
002	2	02	STX	
003	3	03	ETX	Control-C
004	4	04	EOT	
005	5	05	ENQ	
006	6	06	ACK	
007	7	07	BEL	
010	8	08	BS	
011	9	09	HT	tabulation
012	10	0a	LF	saut de ligne
013	11	0b	VT	
014	12	0c	FF	
015	13	0d	CR	retour chariot
016	14	0e	SO	
017	15	0f	SI	
020	16	10	DLE	
021	17	11	DC1	Control-Q
022	18	12	DC2	
023	19	13	DC3	Control-S
024	20	14	DC4	
025	21	15	NAK	
026	22	16	SYN	
027	23	17	ETB	
030	24	18	CAN	
031	25	19	EM	
032	26	1a	SUB	
033	27	1b	ESC	
034	28	1c	FS	

035	29	1d	GS	
036	30	1e	RS	
037	31	1f	US	
040	32	20	espace	
041	33	21	!	point d'exclamation
042	34	22	"	guillemet
043	35	23	#	dièse
044	36	24	$	dollar
045	37	25	%	pour cent
046	38	26	&	et commercial
047	39	27	'	apostrophe
050	40	28	(	parenthèse ouvrante
051	41	29	)	Parenthèse fermante
052	42	2a	*	astérisque
053	43	2b	+	plus
054	44	2c	,	Virgule
055	45	2d	-	tiret
056	46	2e	.	point final
057	47	2f	/	barre oblique (slash)
060	48	30	0	
061	49	31	1	
062	50	32	2	
063	51	33	3	
064	52	34	4	
065	53	35	5	
066	54	36	6	
067	55	37	7	
070	56	38	8	
071	57	39	9	

072	58	3a	:	deux-points
073	59	3b	;	point-virgule
074	60	3c	<	inférieur
075	61	3d	=	Égal
076	62	3e	>	supérieur
077	63	3f	?	Point d'interrogation
100	64	40	@	at
101	65	41	A	
102	66	42	B	
103	67	43	C	
104	68	44	D	
105	69	45	E	
106	70	46	F	
107	71	47	G	
110	72	48	H	
111	73	49	I	
112	74	4a	J	
113	75	4b	K	
114	76	4c	L	
115	77	4d	M	
116	78	4e	N	
117	79	4f	O	
120	80	50	P	
121	81	51	Q	
122	82	52	R	
123	83	53	S	
124	84	54	T	
125	85	55	U	
126	86	56	V	
127	87	57	W	
130	88	58	X	

131	89	59	Y	
132	90	5a	Z	
133	91	5b	[	crochet ouvrant
134	92	5c	\	barre oblique inversée (anti-slash)
135	93	5d	]	crochet fermant
136	94	5e	^	accent circonflexe
137	95	5f	_	souligné
140	96	60	`	
141	97	61	a	
142	98	62	b	
143	99	63	c	
144	100	64	d	
145	101	65	e	
146	102	66	f	
147	103	67	g	
150	104	68	h	
151	105	69	i	
152	106	6a	J	
153	107	6b	k	
154	108	6c	l	
155	109	6d	m	
156	110	6e	n	
157	111	6f	o	
160	112	70	p	
161	113	71	q	
162	114	72	r	
163	115	73	s	
164	116	74	t	
165	117	75	u	
166	118	76	v	
167	119	77	w	

170	120	78	x	
171	121	79	y	
172	122	7a	z	
173	123	7b	{	accolade ouvrante
174	124	7c	|	barre verticale
175	125	7d	}	accolade fermante
176	126	7e	~	tilde
177	127	7f	Effacement	